UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE,FISICHE E NATURALI
CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

UN MODELLO DI AUTOMA CELLULARE PER LA FLUIDODINAMICA TRIDIMENSIONALE

Relatore: Prof. Gianpiero CATTANEO
Correlatore: Prof. Giancarlo MAURI

Tesi di laurea di:
Paolo ZULIANI
Matricola: 421381

Anno Accademico 1996-1997

Indice

Capitolo 1

Introduzione

1.1 Gli automi cellulari

Nel caso più semplice un automa cellulare è costituito da un aggregato di molte componenti identiche, ciascuna delle quali è caratterizzata da uno stato appartenente ad un insieme finito. L'automa evolve attraverso una successione di passi temporali discreti. Ad ogni passo lo stato viene aggiornato secondo una regola che determina il nuovo valore sulla base dei valori posseduti all'istante precedente da un insieme finito di celle. Questo insieme, chiamato *intorno*, comprende generalmente la cella stessa e un certo numero di celle "limitrofe." La regola viene applicata simultaneamente a tutte le celle che compongono l'automa cellulare. Riassumendo gli automi cellulari sono caratterizzati da cinque invarianti fondamentali [FTW84]:

1. Consistono di un reticolo di celle.
2. Evolvono in passi temporali discreti.
3. L'insieme degli stati di una cella è finito e identico per ogni cella.
4. Lo stato di ogni cella evolve in accordo alla stessa regola (*uniformità*).
5. La regola che definisce l'evoluzione dipende solo da un sottoinsieme finito di celle limitrofe (*località*).

Gli automi cellulari costituiscono quindi un modello discreto per la simulazione dei sistemi dinamici complessi caratterizzati da interazioni *uniformi* (4) e *locali* (5), e possono essere visti come un insieme di processori che computano in sincronia la stessa funzione, raccogliendo le informazioni necessarie all'elaborazione solo dalle celle vicine.

Negli ultimi anni gli automi cellulari hanno acquisito una sempre maggiore importanza nel campo della modellazione e della simulazione: in particolare la loro uniformità, la loro località e il loro intrinseco parallelismo li rendono particolarmente semplici da implementare su macchine ad architettura vettoriale o parallela. Malgrado ciò, attualmente, la maggior parte delle tecniche di simulazione basate su automi cellulari non sono in grado di competere, quanto a prestazioni computazionali, con i tradizionali sistemi fondati sulla risoluzione numerica approssimata di equazioni differenziali.

La ragione è molto semplice. Praticamente tutti i supercalcolatori disponibili sono pensati e ottimizzati per operare nell'ambiente del calcolo numerico e quindi, in ultima analisi, per effettuare operazioni su numeri in formato floating-point. Gli automi cellulari, o almeno la maggior parte di essi, funzionano basandosi unicamente su semplici operazioni simboliche implementabili come operazioni su singoli bit. Quindi la grande potenze di calcolo in virgola mobile non può essere sfruttata.

Oltre ai vantaggi computazionali alcune caratteristiche tipiche degli automi cellulari permettono di ottenere un grande guadagno rispetto ai metodi classici. Uno dei più evidenti è l'enorme aumento della risoluzione delle simulazioni. Ogni cella di un automa cellulare è generalmente molto piccola rispetto al totale del sistema e quindi, globalmente, si ottiene una rappresentazione dotata di un dettaglio molto elevato, irraggiungibile con altri mezzi.

1.2 Automi cellulari e fluidodinamica

Uno dei primi settori in cui gli automi cellulari hanno trovato spazio come mezzo di simulazione è la fluidodinamica computazionale. Questo settore è stato dominato fino a non molto tempo addietro dalle tecniche di risoluzione numerica basate sulla discretizzazione e risoluzione approssimata delle equazioni di Navier-Stokes. Questi metodi presentano però alcune difficoltà di rilievo

Uno dei principali problemi di queste tecniche è la definizione delle condizioni al contorno. La descrizione di alto livello che si ha del fluido obbliga ad utilizzare sistemi per la definizione delle zone solide molto complessi, senza i quali non è possibile raggiungere una adeguata precisione. Malgrado la sofisticazione a cui sono giunte, queste tecniche si trovano comunque in enorme difficoltà nel caso in cui si debbano simulare condizioni molto complesse e irregolari quali quelle che si possono trovare per esempio nei materiali porosi. Inoltre, in molti casi si possono verificare fenomeni di in-

stabilità numerica dovuti alla presenza di singolarità nelle stesse equazioni di Navier-Stokes, fenomeni che possono inficiare gravemente la precisione della simulazione o addirittura provocare il blocco dell'esecuzione a causa dei problemi di overflow.

L'approccio basato su automi cellulari parte invece dall'idea di simulare, per quanto possibile, quello che avviene a livello molecolare nei fluidi reali. Le singole celle sono generalmente molto semplici e ne occorrono ovviamente un gran numero per poter ottenere una approssimazione valida di ciò che succede nella realtà. Questo, se da un lato aumenta le richieste computazionali del sistema, dall'altro permette di approssimare con grande precisione le parti solide. Difatti, è sufficiente marcare in modo opportuno le celle corrispondenti alle zone solide e introdurre una regola di aggiornamento estesa per ottenere la realizzazione delle condizioni al contorno.

La rappresentazione a livello molecolare del comportamento del fluido permette anche l'eventuale introduzione di reazioni chimiche o la rappresentazione contemporanea di più fluidi differenti, sia miscibili che immiscibili, con conseguente enorme estensione delle possibilità operative.

In ogni caso, essendo l'evoluzione temporale basata esclusivamente su operazioni subsimboliche, ossia spostamento e manipolazione di particelle, sono completamente eliminati i problemi relativi all' instabilità numerica.

1.2.1 Il modello HPP

Il primo automa cellulare introdotto nel campo della fluidodinamica è il modello HPP proposto nel 1976 da Hardy, de Pazzis e Pomeau [HdPP76]. Questo modello è costituito da un reticolo bidimensionale quadrato in cui ogni cella è connessa con le celle adiacenti lungo le quattro direzioni cardinali. Ognuna di queste celle può contenere fino a quattro particelle, una per ciascuna delle quattro direzioni possibili (vedi figura 1.1). Un principio di esclusione vieta la presenza contemporanea di più di una particella in una direzione.

Ogni particella ha massa unitaria e velocità unitaria, diretta secondo il sito cui appartiene e con verso che punta alla cella adiacente. Per comodità l'evoluzione temporale dell'automa cellulare viene suddivisa in due sottopassi: passo di propagazione e passo di collisione.

Durante il passo di propagazione ogni particella si muove secondo la propria direzione spostandosi nella cella adiacente. Se si considera la velocità unitaria, la distanza tra le celle unitaria e la durata del passo di propagazione unitario si ha una perfetta corrispondenza col caso fisico di particelle in moto rettilineo uniforme.

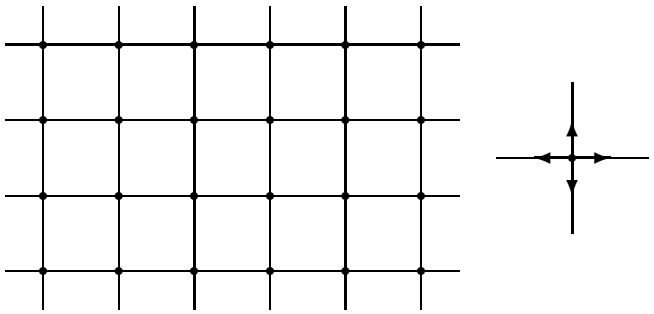

Figura 1.1: Il reticolo del modello HPP e la cella corrispondente. Ogni nodo è indicato con un pallino. Le frecce indicano le direzioni in cui possono essere presenti le particelle

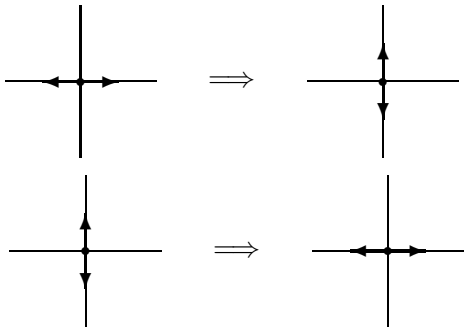

Figura 1.2: Le due configurazioni di collisione lecite nel modello HPP

Durante il passo di collisione le particelle all'interno di una cella interagiscono unicamente tra di loro. Imponendo come condizione necessaria per l'interazione il rispetto delle leggi base della fisica, ossia conservazione della massa e della quantità di moto, si può verificare come esistano due sole configurazioni che danno origine a collisioni lecite (vedi figura 1.2). Inoltre, dato che la velocità delle particelle non cambia, si ha anche conservazione dell' energia.

Globalmente quindi abbiamo una rappresentazione approssimata di quello che succede in un fluido reale, con spostamenti e collisioni guidati dalle leggi della meccanica classica. Esperimenti compiuti con questi sistemi hanno mostrato come, malgrado la semplicità e la notevole imprecisione della approssimazione, si possa ottenere lo sviluppo di alcuni fenomeni fisici pre-

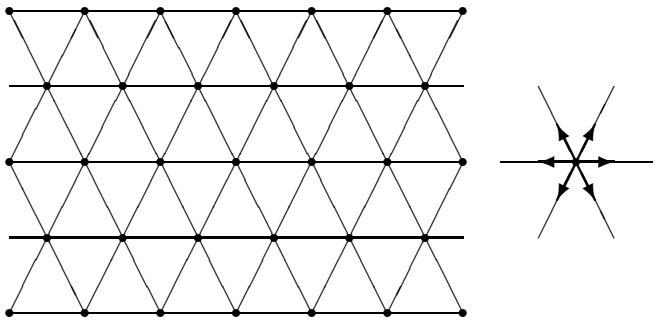

Figura 1.3: Il reticolo e la cella nel modello FHP

senti nei fluidi reali. Per esempio è stato possibile dimostrare che la velocità del suono è isotropa e quindi, malgrado il sistema non presenti simmetria circolare, si possono ottenere onde con propagazione circolare.

Il modello HPP risulta comunque completamente inadatto alla realizzazione di simulazioni. Esso non soddisfa nemmeno le equazioni di Navier-Stokes ed è quindi impensabile utilizzarlo nel campo nella fluidodinamica computazionale, dove è richiesta una grande precisione. Malgrado ciò un modello solo leggermente più complesso permette di raggiungere lo scopo.

1.2.2 Il modello FHP

Il modello FHP è stato introdotto nel 1986 da Frisch, Hasslacher e Pomeau e costituisce una semplice variante rispetto al modello HPP [FHP88]. Esso si basa su di un reticolo esagonale (o triangolare a seconda dei punti di vista) e di conseguenza le celle contengono fino a 6 particelle (vedi figura 1.3).

L'evoluzione dell'automa è del tutto analoga a quella del modello HPP, con identica suddivisione in passo di propagazione e passo di collisione realizzati basandosi sulle leggi della meccanica classica. Le collisioni, in questo caso, risultano molto più complesse e possono avvenire con la presenza di due, tre oppure quattro particelle. Due esempi sono riportati in figura 1.4. Inoltre, alcune configurazioni iniziali possono avere più configurazioni di collisione possibili. In queste situazioni si usa scegliere casualmente, direttamente durante l'evoluzione e con probabilità uniforme, una qualunque delle configurazioni di collisione possibili (vedi figura 1.5).

La differenza fondamentale risiede nel fluido macroscopico risultante dall'evoluzione. Difatti, sotto opportune ipotesi, è possibile dimostrare che il sistema soddisfa le equazioni di Navier-Stokes per il caso di fluido incompri-

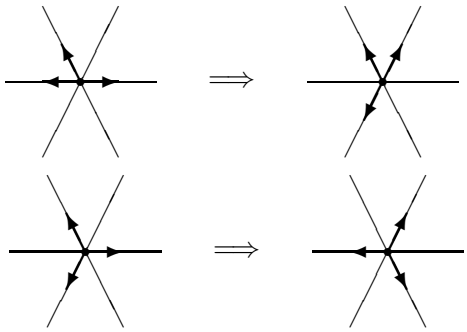

Figura 1.4: Due possibili collisioni nel modello FHP

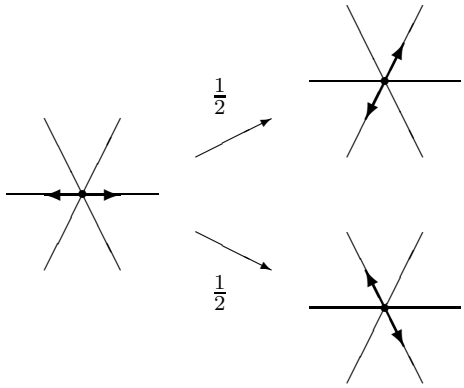

Figura 1.5: Un esempio di configurazione con più di una collisione possibile. La scelta avviene casualmente con probabilità uniforme

mibile e quindi risulta adatto all' implementazione di sistemi di simulazione per la fluidodinamica [Wol86].

1.2.3 I modelli tridimensionali

L'estensione al tridimensionale dei modelli ad automi cellulari presenta non poche difficoltà. Nelle tre dimensioni non esiste alcun solido che riempia lo spazio e che soddisfi le condizioni sull' isotropia dei tensori che permettono di ottenere le equazioni di Navier-Stokes a partire dalla dinamica microscopica del sistema [Wol86]. Per ottenere il risultato si è costretti a passare alle quattro dimensioni, basandosi su di un reticolo ipercubico e provvedendo poi a proiettare l'automa risultante sulle tre dimensioni. Questo modello si chiama FCHC (Face Centered HyperCubic).

La cella FCHC contiene fino a 24 particelle e di conseguenza l' implementazione risulta problematica, in particolare per quel che riguarda il passo di collisione in quanto il numero di collisioni diventa enorme, dell'ordine di grandezza del numero di configurazioni possibili (2^{24}).

Il modello rombododecedrico oggetto della tesi è invece un modello di automa cellulare tridimensionale che a prima vista non sembrerebbe molto interessante. Infatti il tensore di quarto ordine è anisotropo, pregiudicando quindi la possibilità di soddisfare le equazioni di Navier-Stokes. Nonostante ciò il modello merita attenzione per alcuni motivi: l'anisotropia del tensore di quarto ordine si ripercute sulle equazioni di Navier-Stokes sotto forma di una perturbazione simmetrica in due direzioni ortogonali. Tale perturbazione include solo termini differenziali di ordine due e quadratici, quindi in prima approssimazione trascurabili.

Inoltre, essendo un modello a 12 particelle, la complessità computazionale della simulazione non raggiunge livelli proibitivi, permettendo quindi una effettiva utilizzazione dell'automa nel campo della fluidodinamica computazionale.

1.3 Limitazioni

Esistono alcuni problemi legati al numero limitato di direzioni presenti nel reticolo. Questo provoca l'emergere di alcuni fenomeni non fisici quali una variazione anisotropica della viscosità in funzione della velocità e una anomala dipendenza della pressione dalla velocità. Di conseguenza si hanno delle forti limitazioni sul range di velocità utilizzabili nelle simulazioni, pena un forte scadimento nella precisione dei risultati.

A questo si aggiunge la presenza di rumore statistico nell'evoluzione del sistema. Per eliminare questo problema si è costretti a mediare i risultati su periodi molto lunghi con la conseguente eliminazione di eventuali fenomeni transitori che potrebbero essere di notevole interesse per la simulazione.

In ogni caso i modelli basati su automi cellulari sono già in grado di competere con i metodi classici nel caso della percolazione, in alcuni casi in cui sia necessario rappresentare più fluidi di tipo diverso sia miscibili che immiscibili e nel settore della simulazione di fluidi in presenza di reazioni chimiche.

Capitolo 2

Il modello rombododecaedrico

2.1 Introduzione

In questo capitolo verrà presentato il modello studiato: il modello rombododecaedrico. In particolare, dopo aver illustrato la costruzione del reticolo tridimensionale, si passerà alla spiegazione delle proprietà geometriche e analitiche del modello.

2.2 Costruzione del reticolo

Il modello proposto si basa su un poliedro irregolare, il *rombododecaedro*, che è in grado di riempire lo spazio tridimensionale senza lasciare cavità. Il rombododecaedro è un poliedro composto da dodici facce rombiche uguali e può essere costruito poggiando su ogni faccia di un cubo con spigoli di lunghezza unitaria, una piramide regolare a base quadrata di lato unitario e altezza mezza unità. Da questa costruzione si ha che le facce oblique adiacenti di due piramidi giacciono sullo stesso piano formando un rombo, ovvero una faccia del rombododecaedro. La figura 2.1 illustra la costruzione del solido. Per riempire lo spazio è sufficente appoggiare su ogni faccia del rombododecaedro un altro rombododecaedro. In questo modo viene a crearsi una struttura tridimensionale che ha nel suo primo vicinato dodici altre strutture. Nello spazio non si creano buchi perchè le piramidi hanno esattamente un sesto del volume del cubo di partenza e base uguale alle altre facce[Sal94]. Nelle figure 2.2 e 2.3 possiamo vedere alcuni esempi di incastri a due, tre e quattro solidi.

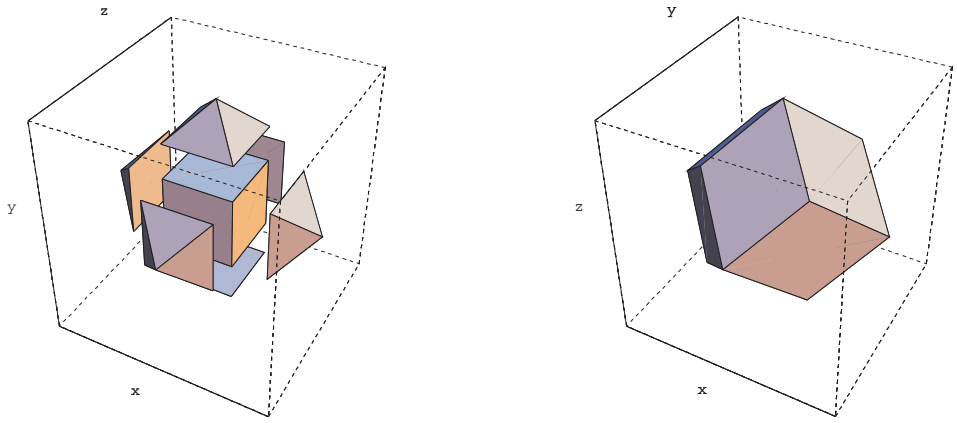

Figura 2.1: costruzione del rombododecaedro

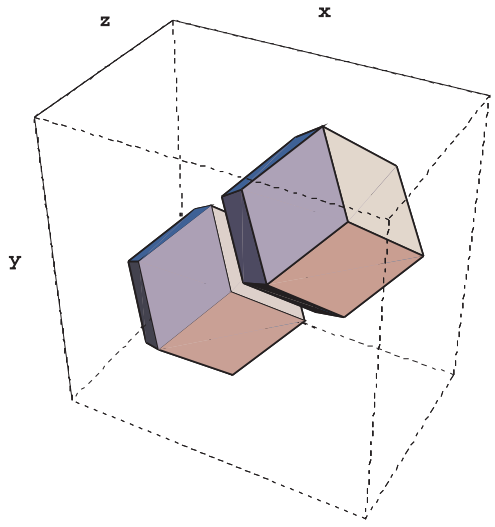

Figura 2.2: costruzione del reticolo (1)

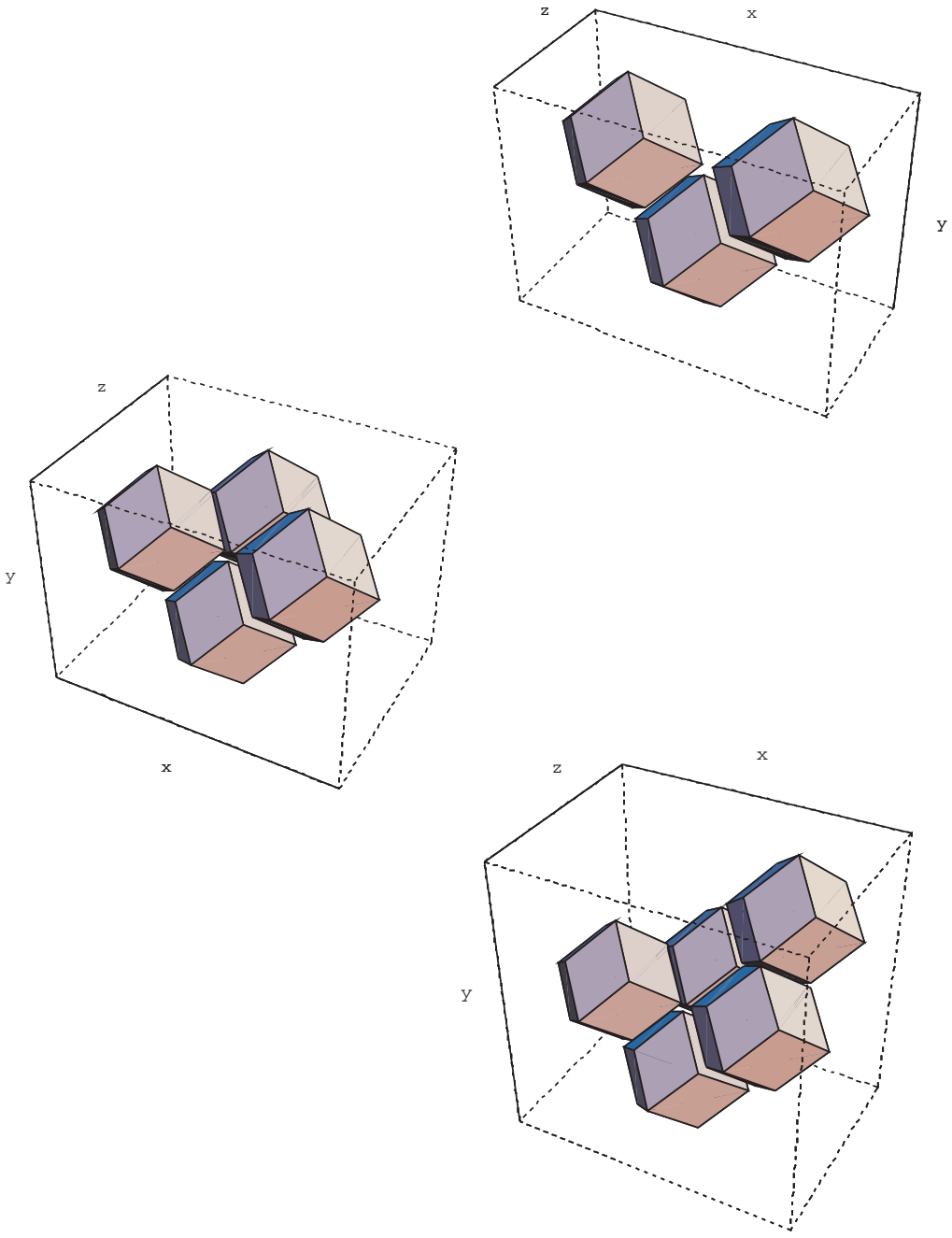

Figura 2.3: costruzione del reticolo (2)

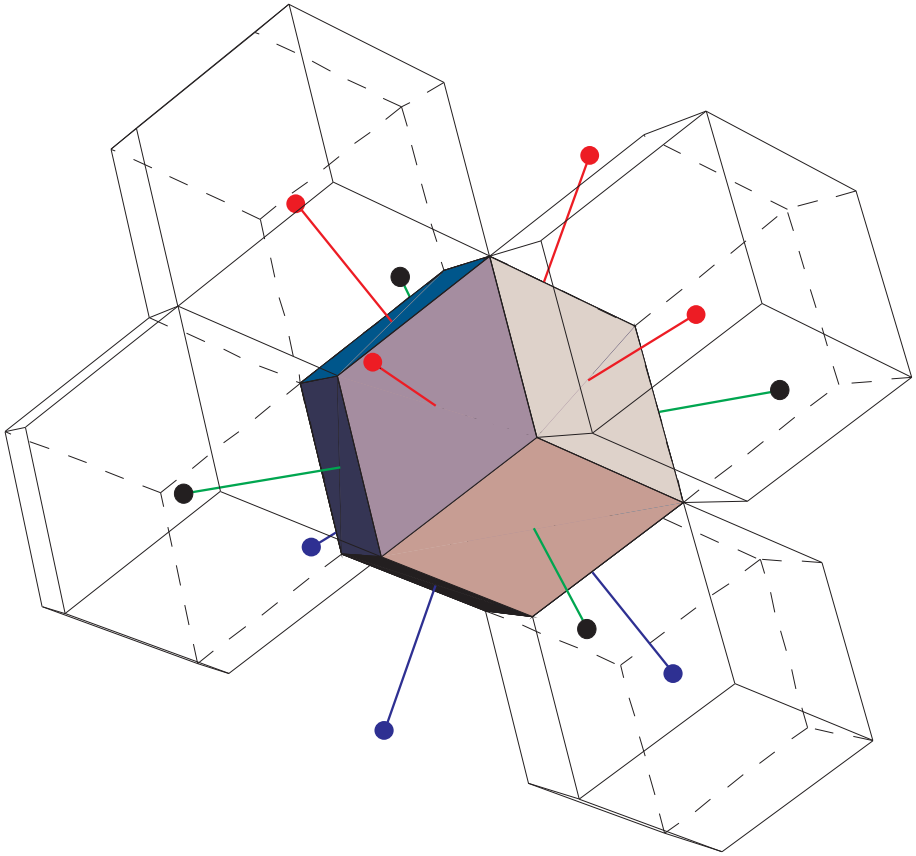

Figura 2.4: vettori di collegamento

In una struttura così ottenuta si ha che la distanza tra il centro di un rombododecaedro e il centro dei suoi dodici vicini è identica. Possiamo perciò considerare come possibili direzioni della velocità di una particella le direzioni dei dodici vettori che si ottengono congiungendo il centro di un rombododecaedro con il centro dei sui primi vicini. La figura 2.4 mostra un rombododecaedro e i vettori che lo collegano al centro dei suoi vicini.

A questo punto possiamo pensare di sostituire ogni rombodocaedro della struttura tridimensionale con il suo centro: otterremo così una discretizzazione dello spazio tridimensionale in cui le direzioni ammesse sono solo dodici. Da qui in avanti chiameremo *nodo* il centro di un rombododecaedro. Quindi si può pensare il fluido come un insieme di particelle di massa unitaria libere di muoversi lungo i vettori che collegano nodi del reticolo, ovvero i centri dei rombododecaedri. Quindi per ogni nodo esisteranno dodici tipi

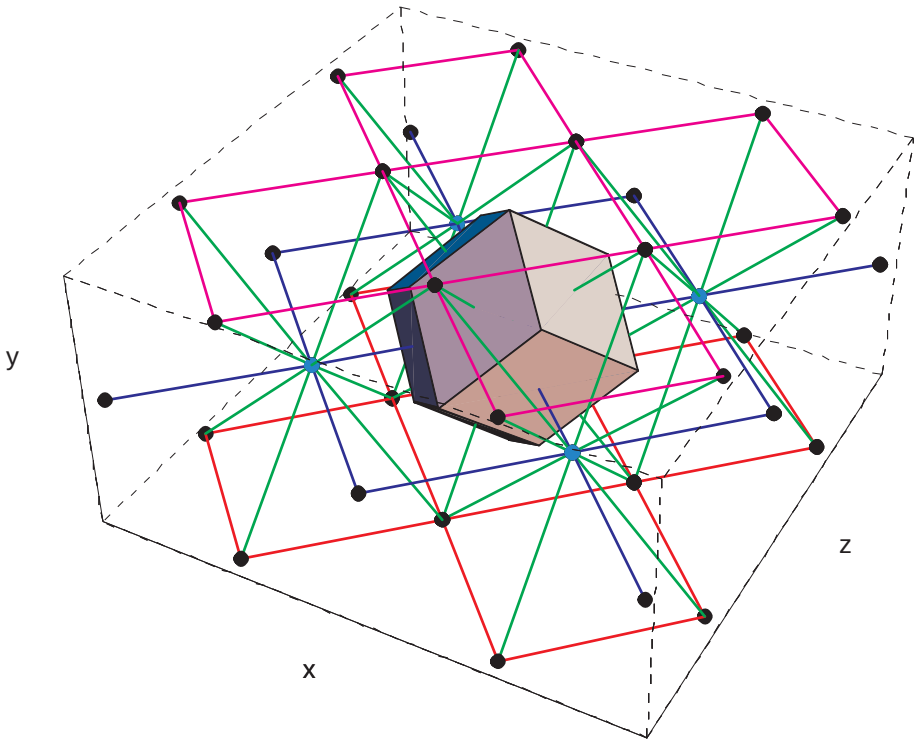

Figura 2.5: esempio di reticolo

di particelle, una per ogni direzione ammessa. La figura 2.5 dà un esempio di reticolo, lasciando un solo rombododecaedro per comodità. Nella teoria degli automi cellulari si ritiene valido un importante *principio di esclusione* che può essere così enunciato [FHP88]:

> *ogni nodo del reticolo può essere occupato al più da una sola particella per ogni specie.*

In conseguenza a questo principio ogni nodo del reticolo può assumere solo uno dei $2^{12} = 4096$ stati possibili.

2.3 Formalizzazione del modello

L'intorno di un generico nodo $\vec{r}$ del reticolo è definito ricorrendo a dodici vettori $\vec{e_i}$ per $i \in B = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$, nel seguente modo:

$$V(\vec{r}) = \{\vec{r} + \vec{e_i} \ : \ i \in B\}$$

La figura 2.6 visualizza l'intorno $V(\vec{r})$:

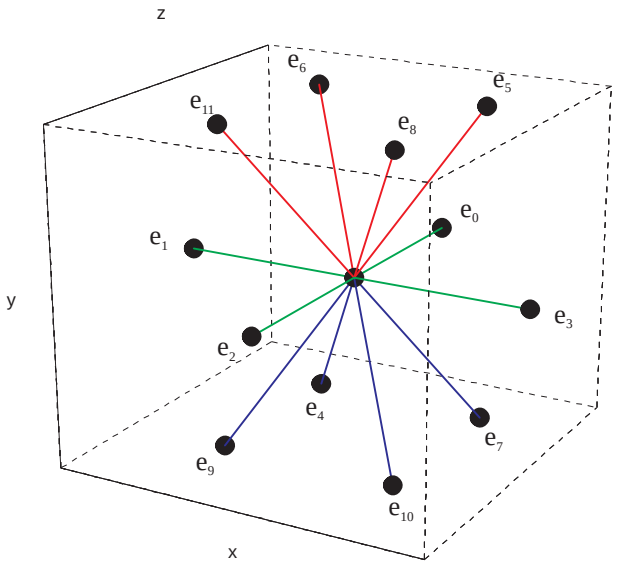

Figura 2.6: intorno di un nodo

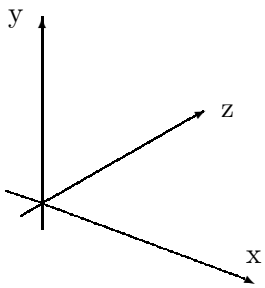

Figura 2.7: sistema di riferimento

Nel sistema di riferimento cartesiano $Oxyz$ di figura 2.7, in cui l'origine coincide con il centro del rombododecaedro, il set of velocity vectors ha le seguenti coordinate:

$$\begin{array}{rcl}
\vec{e_0} &=& (\ \ 0, \quad\ 0, \quad\ 1\ \ ) \\
\vec{e_1} &=& (\ -1, \quad\ 0, \quad\ 0\ \ ) \\
\vec{e_2} &=& (\ \ 0, \quad\ 0, \quad -1\ ) \\
\vec{e_3} &=& (\ \ 1, \quad\ 0, \quad\ 0\ \ ) \\
\vec{e_4} &=& (\ -1/2, \ -1/\sqrt{2}, \ 1/2\ ) \\
\vec{e_5} &=& (\ \ 1/2, \ \ 1/\sqrt{2}, \ 1/2\ ) \\
\vec{e_6} &=& (\ -1/2, \ \ 1/\sqrt{2}, \ 1/2\ ) \\
\vec{e_7} &=& (\ \ 1/2, \ -1/\sqrt{2}, \ 1/2\ ) \\
\vec{e_8} &=& (\ \ 1/2, \ \ 1/\sqrt{2}, \ -1/2\ ) \\
\vec{e_9} &=& (\ -1/2, \ -1/\sqrt{2}, \ -1/2\ ) \\
\vec{e_{10}} &=& (\ \ 1/2, \ -1/\sqrt{2}, \ -1/2\ ) \\
\vec{e_{11}} &=& (\ -1/2, \ \ 1/\sqrt{2}, \ -1/2\ )
\end{array}$$

La scelta di tale sistema di riferimento non è casuale, ma basata su una osservazione sulle equazioni di Navier-Stokes che verrà presentata più avanti in questo capitolo.

2.4 Proprietà geometriche

Introduciamo innanzitutto il concetto di *isometria*: una trasformazione dello spazio che lascia invariate le distanze. In maniera più formale è possibile dare la seguente:

Definizione. Sia (X,d) uno spazio metrico. Una applicazione $g : X \to X$ è detta **isometria** sse:

$$\forall\, x, y \in X \ \Rightarrow\ d(g(x), g(y)) = d(x, y)$$

Nella famiglia di spazi metrici (R^n, d) dove d è la metrica euclidea, si può facilmente dimostrare che due semplici classi di applicazioni soddisfano la definizione data:

- le permutazioni di coordinate;
- i ribaltamenti di coordinate (cambiamenti di segno);

Enunciamo quindi le seguenti:

Proposizione 2.4.1. *Negli spazi metrici euclidei (R^n, d) le permutazioni di coordinate sono isometrie.*

Dimostrazione. Siano $\vec{x} = (x_1, \ldots, x_n)$ e $\vec{y} = (y_1, \ldots, y_n)$ due vettori di R^n. Come è noto la loro distanza è:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

La stessa permutazione di coordinate di $\vec{x}$ e $\vec{y}$ si traduce in un diverso ordine degli addendi della sommatoria. Quindi per la commutatività della somma la proposizione è dimostrata. □

Proposizione 2.4.2. *Negli spazi metrici euclidei (R^n, d) i ribaltamenti di coordinate sono isometrie.*

Dimostrazione. Siano $\vec{x} = (x_1, \ldots, x_n)$ e $\vec{y} = (y_1, \ldots, y_n)$ due vettori di R^n. La loro distanza è:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Per dimostare questa proposizione basta notare che:

$$((-a) - (-b))^2 = (b - a)^2 = (a - b)^2$$

Quindi applicando la stesso cambiamento di segno alle coordinate di $\vec{x}$ e $\vec{y}$ i termini nella sommatoria rimarranno invariati. □

Le isometrie con la usuale operazione di composizione di applicazioni formano un *semigruppo*. Questo fatto è dimostrato nella:

Proposizione 2.4.3. *Sia $I_d(X)$ l'insieme di tutte le isometrie sullo spazio metrico (X, d). Sia $\circ$ l'operazione di composizione di applicazioni. Allora la struttura $(I_d(X), \circ)$ è un semigruppo.*

Dimostrazione. Bisogna dimostrare che $\circ$ è una operazione associativa e chiusa su $I_d(X)$. Come è ben noto la composizione di applicazioni è associativa, quindi resta da dimostare la chiusura di $\circ$, ovvero che la composizione di due isometrie è ancora una isometria.

Siano $f, g \in I_d(X)$ e $x, y \in X$ allora, per la definizione di composizione di applicazioni:

$$d(f \circ g(x), f \circ g(y)) = d(f(g(x)), f(g(y)))$$

ma f e g sono isometrie, quindi:

$$d(f(g(x)), f(g(y))) = d(g(x), g(y)) = d(x, y)$$

□

Dalle precedenti proposizioni si può enunciare il seguente:

Corollario. *Nello spazio metrico euclideo (R^n, d) le permutazioni e i ribaltamenti di coordinate sono isometrie.*

Dimostrazione. Per le proposizioni 2.4.1 e 2.4.2 le permutazioni di coordinate e i ribaltamenti sono delle isometrie. Per la proposizione 2.4.3 la loro composizione è ancora una isometria. □

A questo punto possiamo individuare quali sono le isometrie che lasciano invariante i vettori velocità del modello rombododecaedrico.

Proposizione 2.4.4. *Le applicazioni che ribaltano la componente y e permutano e ribaltano le componenti x e z sono isometrie che lasciano invariante il set of velocity vectors.*

Dimostrazione. Per il corollario precedente esse sono isometrie. Inoltre analizzando la struttura dei vettori si può notare che per le componenti x e z esistono tutte le possibili permutazioni e ribaltamenti; per la componente y esiste il solo ribaltamento. □

Esistono altre isometrie, generate da rotazioni rigide del sistema di riferimento mantenenti l'ortogonalità tra gli assi (altrimenti non si hanno più isometrie). Tra queste quelle che interessano sono le isometrie che mantengono invariato l'insieme dei vettori del modello, ovvero rotazioni multiple di $\pi/2$ rad. Quindi l'insieme delle isometrie cercato è formato da tali rotazioni rigide e dalle isometrie elencate nella proposizione 2.4.4. Tutto ciò permette di enunciare due importanti proposizioni.

Proposizione 2.4.5. *L'insieme delle isometrie che lascia invariante l'insieme dei velocity vectors è un gruppo con la composizione di applicazioni.*

Dimostrazione. La proposizione 2.4.3 dimostra che una tale struttura è un semigruppo. Dobbiamo quindi dimostrare che esiste l'unità ed ogni elemento è invertibile.

L'unità è banalmente l'applicazione identica i (che è anche una isometria sotto qualunque metrica).

Per l'inverso distinguiamo le isometrie generate da rotazioni da quelle generate da permutazioni e ribaltamenti delle coordinate. Nel primo caso data una qualsiasi rotazione ne posso costruire subito un'altra in senso opposto. Nel secondo caso l'inverso è dato dall'isometria stessa: infatti applicando la stessa permutazione delle componenti x e z ad un vettore si riottiene il vettore di partenza; inoltre cambiando il segno due volte si ritorna al segno originario. □

La seguente proposizione afferma che i vettori del velocity set sono in un certo senso interscambiabili.

Proposizione 2.4.6. *Sia V l'insieme dei velocity vectors del modello. Per ogni $(\vec{e}_i, \vec{e}_j) \in V$ esiste una isometria che mappa $\vec{e}_i$ in $\vec{e}_j$.*

Dimostrazione. Possiamo procedere per costruzione: se $i, j \in \{0, 1, 2, 3\}$ allora possiamo usare permutazioni e ribaltamenti delle coordinate x, z. Se $i, j \in \{4, \ldots, 11\}$ si applicano ribaltamenti delle componenti x, y e z. In tutti gli altri casi si usano rotazioni rigide di $\pi/2$ rad dell'intero sistema di riferimento combinate con permutazioni e ribaltamenti. □

Per chiarire diamo qualche esempio del procedimento:

Esempio 2.4.1. Calcolare l'isometria che mappa $\vec{e}_7$ in $\vec{e}_8$: basta ribaltare le coordinate y e z.

Esempio 2.4.2. Calcolare l'isometria che mappa $\vec{e}_7$ in $\vec{e}_{11}$: dalla figura 2.6 si vede che questi due vettori sono opposti, quindi si devono ribaltare tutte le coordinate. Questa isometria corrisponde ad una simmetria rispetto al piano $y = 0$.

Esempio 2.4.3. Calcolare l'isometria che mappa $\vec{e}_0$ in $\vec{e}_{11}$. Procediamo in due passi: con una rotazione rigida di $\pi/2$ rad nel verso dell'osservatore del sistema di riferimento otteniamo che il vettore $\vec{e}_0$ prende il posto del vettore $\vec{e}_5$. Questa rotazione si può formalizzare nella matrice (in appendice verrà dato un richiamo sulla costruzione delle matrici di rotazione [Rut89]):

$$M = \begin{pmatrix} 1/2 & 1/\sqrt{2} & 1/2 \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 1/2 & -1/\sqrt{2} & 1/2 \end{pmatrix}$$

La figura 2.8 illustra la nuova posizione dei vettori a seguito della rotazione.

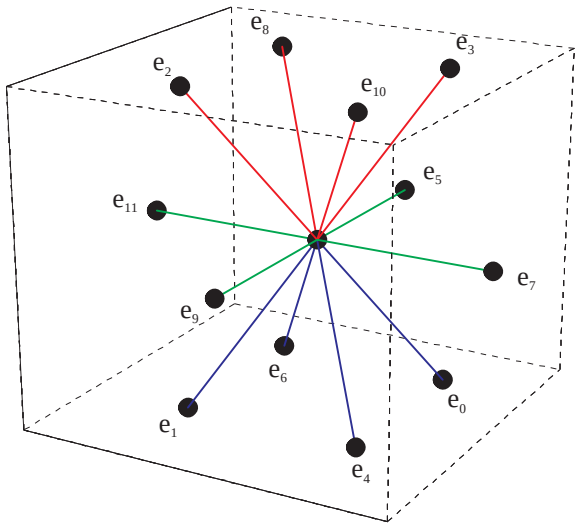

Figura 2.8: rotazione del sistema di riferimento

Applicando la matrice a tutti i velocity vectors abbiamo:

$$\begin{array}{llll} M\cdot\vec{e}_0 & =\vec{e}_5 & M\cdot\vec{e}_6 & =\vec{e}_8 \\ M\cdot\vec{e}_1 & =\vec{e}_{11} & M\cdot\vec{e}_7 & =\vec{e}_0 \\ M\cdot\vec{e}_2 & =\vec{e}_9 & M\cdot\vec{e}_8 & =\vec{e}_{10} \\ M\cdot\vec{e}_3 & =\vec{e}_7 & M\cdot\vec{e}_9 & =\vec{e}_1 \\ M\cdot\vec{e}_4 & =\vec{e}_6 & M\cdot\vec{e}_{10} & =\vec{e}_4 \\ M\cdot\vec{e}_5 & =\vec{e}_3 & M\cdot\vec{e}_{11} & =\vec{e}_2 \end{array}$$

Come si vede $\vec{e}_0$ è mappato in $\vec{e}_5$. Adesso ribaltiamo le componenti x e z e otterremo il vettore $\vec{e}_{11}$. Anche quest'ultima operazione può essere facilmente formalizzata con una matrice:

$$N = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

Quindi l'isometria richiesta è data dalla matrice P così definita:

$$P = N\cdot M = \begin{pmatrix} -1/2 & -1/\sqrt{2} & -1/2 \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} \\ -1/2 & 1/\sqrt{2} & -1/2 \end{pmatrix}$$

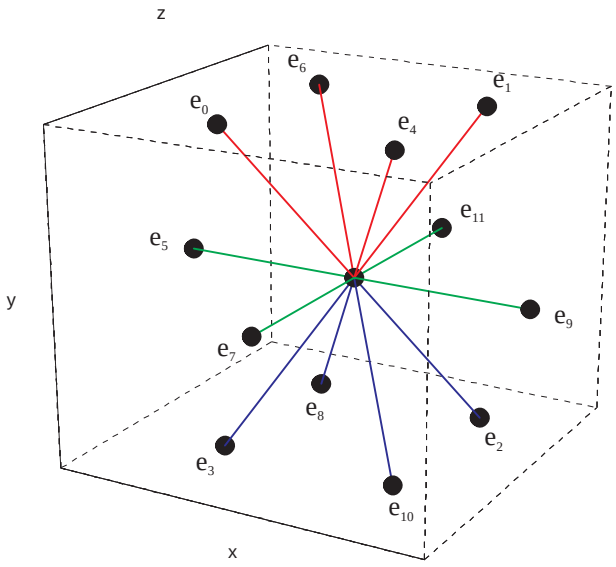

Figura 2.9: configurazione finale

Ora applichiamo P ai vettori velocità:

$$\begin{array}{llll} P\cdot\vec{e}_0 & =\vec{e}_{11} & P\cdot\vec{e}_6 & =\vec{e}_6 \\ P\cdot\vec{e}_1 & =\vec{e}_5 & P\cdot\vec{e}_7 & =\vec{e}_2 \\ P\cdot\vec{e}_2 & =\vec{e}_7 & P\cdot\vec{e}_8 & =\vec{e}_4 \\ P\cdot\vec{e}_3 & =\vec{e}_9 & P\cdot\vec{e}_9 & =\vec{e}_3 \\ P\cdot\vec{e}_4 & =\vec{e}_8 & P\cdot\vec{e}_{10} & =\vec{e}_{10} \\ P\cdot\vec{e}_5 & =\vec{e}_1 & P\cdot\vec{e}_{11} & =\vec{e}_0 \end{array}$$

Come previsto $\vec{e}_0$ è mappato in $\vec{e}_{11}$ (notare anche l'invarianza dei vettori $\vec{e}_6$ ed $\vec{e}_{10}$). La figura 2.9 illustra la configurazione finale dei vettori velocità.

Nota. La proposizione 2.4.3 può essere estesa se si considera come spazio metrico lo spazio euclideo (R^n, d) e come isometrie *solo* le permutazioni di coordinate: in questo caso si arriva a dimostrare che la struttura ottenuta è un *gruppo*.

Proposizione 2.4.7. *Sia $Q_d(R^n)$ l'insieme delle isometrie ottenute attraverso permutazioni di coordinate nello spazio metrico euclideo (R^n, d). Sia $\circ$ l'operazione di composizione di applicazioni. Allora la struttura $(Q_d(R^n), \circ)$ è un gruppo.*

Dimostrazione. La proposizione 2.4.3 ha già dimostrato che una tale struttura è un semigruppo. Dobbiamo solo dimostrare che esiste l'unità e che ogni elemento è invertibile.

L'unità è l'applicazione identica i. Sia $f \in Q_d(R^n)$ allora il suo inverso f^{-1} esiste sempre dato che si può vedere f come una funzione biunivoca $f : X \to X$, dove $X = \{1, \ldots, n\}$. □

2.5 Equazioni di Navier-Stokes

È stato dimostrato [Sal94] che il modello rombododecaedrico soddisfa una particolare forma perturbata delle equazioni di Navier-Stokes tridimensionali:

$$\begin{aligned}\partial_t(nu_1) + \mu n(\nabla \cdot \vec{u})u_1 = -\partial_1 p + \eta\nabla^2 u_1 + \left(\zeta + \frac{1}{3}\eta\right)\partial_1(\nabla \cdot \vec{u}) + \\ + \frac{n}{4}\mu\left[\partial_1\left({u_1}^2 + {u_2}^2\right) - 2\partial_2(u_1u_2)\right] - \frac{1}{2}\eta(\partial_{11}u_1 + \partial_{22}u_1 + 2\partial_{12}u_2);\end{aligned} \tag{2.1}$$

$$\begin{aligned}\partial_t(nu_2) + \mu n(\nabla \cdot \vec{u})u_2 = -\partial_2 p + \eta\nabla^2 u_2 + \left(\zeta + \frac{1}{3}\eta\right)\partial_2(\nabla \cdot \vec{u}) + \\ + \frac{n}{4}\mu\left[\partial_2\left({u_1}^2 + {u_2}^2\right) - 2\partial_1(u_1u_2)\right] - \frac{1}{2}\eta(\partial_{11}u_2 + \partial_{22}u_2 + 2\partial_{12}u_1);\end{aligned} \tag{2.2}$$

$$\begin{aligned}\partial_t(nu_3) + \mu n(\nabla \cdot \vec{u})u_3 = -\partial_3 p + \eta\nabla_2 u_3 + \left(\zeta + \frac{1}{3}\eta\right)\partial_3(\nabla \cdot \vec{u}) + \\ + \frac{n}{2}\mu\left[\partial_3 {u_3}^2\right] - \eta(\partial_{33}u_3).\end{aligned} \tag{2.3}$$

Dove:

- $\vec{u}$ è la velocità del fluido;
- n è la densità;
- p è la pressione;
- μ è il coefficiente convettivo;
- η è la shear viscosity;
- ζ è la bulk viscosity;

Confrontiamo ora questa forma delle equazioni con le corrette Navier-Stokes, ove i simboli hanno lo stesso precedente significato:

$$\partial_t(n\vec{u}) + \mu n(\nabla \cdot \vec{u})\vec{u} = -\nabla p + \eta\nabla^2\vec{u} + \left(\zeta + \frac{1}{3}\eta\right)\nabla(\nabla \cdot \vec{u}) \tag{2.4}$$

Si nota subito la perturbazione simmetrica presente nelle equazioni (2.1) e (2.2) e la perturbazione asimmetrica nella eq. (2.3). A prima vista potrebbero sembrare dei gravi difetti ma osservando bene le perturbazioni si vede che comprendono termini quadratici rispetto alle componenti di u e termini differenziali di ordine due. Ciò, unito al fatto che le ordinarie eq. di Navier-Stokes sono derivate nell'ipotesi che $|\vec{u}| \ll 1$, rafforza l'effettiva validità del modello rombododecaedrico.

Inoltre introducendo l'ipotesi di incompressibilità del fluido:

$$\nabla \cdot \vec{u} = 0$$

è possibile riscrivere parte della perturbazione della eq. (2.3). L'ipotesi di incompressibilità equivale a:

$$\partial_1 u_1 + \partial_2 u_2 + \partial_3 u_3 = 0$$

quindi:

$$\partial_3 u_3 = -\partial_1 u_1 - \partial_2 u_2$$

Tenendo presente la precedente relazione e l'ipotesi di incompressibilità possiamo riscrivere le eq. idrodinamiche per il modello rombododecaedrico:

$$\begin{aligned}\partial_t(nu_1) = &-\partial_1 p + \eta\nabla^2 u_1 + \frac{n}{4}\mu\left[\partial_1\left({u_1}^2 + {u_2}^2\right) - 2\partial_2(u_1u_2)\right] + \\ &- \frac{1}{2}\eta(\partial_{11}u_1 + \partial_{22}u_1 + 2\partial_{12}u_2);\end{aligned} \tag{2.5}$$

$$\begin{aligned}\partial_t(nu_2) = &-\partial_2 p + \eta\nabla^2 u_2 + \frac{n}{4}\mu\left[\partial_2\left({u_1}^2 + {u_2}^2\right) - 2\partial_1(u_1u_2)\right] + \\ &- \frac{1}{2}\eta(\partial_{11}u_2 + \partial_{22}u_2 + 2\partial_{12}u_1);\end{aligned} \tag{2.6}$$

$$\partial_t(nu_3) = -\partial_3 p + \eta\nabla^2 u_3 + \frac{n}{2}\mu\left[\partial_3 {u_3}^2\right] - \eta(-\partial_{31}u_1 - \partial_{32}u_2). \tag{2.7}$$

Confrontiamole con le corrette eq. di Navier-Stokes per un fluido incompressibile:

$$\partial_t(n\vec{u}) = -\nabla p + \eta\nabla^2\vec{u} \tag{2.8}$$

Si vede che, a parte il termine differenziale di primo ordine dell'eq. (2.7), si è riusciti a confinare le perturbazioni nelle due componenti u_1 e u_2. Quindi

per fluidi incomprimibili con componenti u_1 e u_2 piccole ci si può aspettare in generale un buon comportamento del modello. Questo può quasi sempre essere assicurato facendo coincidere la direzione di scorrimento del fluido con l'asse z. Nell'implementazione del software di simulazione è stato perciò tenuto presente questo fatto.

Capitolo 3

Coefficienti di trasporto

3.1 Introduzione

In questo capitolo verrà introdotto un metodo per la valutazione approssimata dei coefficienti di trasporto. Come esempio di applicazione del metodo si calcolerà la viscosità di un generico automa cellulare n-dimensionale con collisioni esclusivamente a due particelle. Successivamente si passerà alla valutazione dei coefficienti di trasporto per il modello rombododecaedrico.

3.2 Fondamenti teorici

Riassumiamo qui brevemente le basi teoriche che portano alla determinazione dei coefficienti di trasporto. Un insieme di configurazioni microscopiche può essere descritto da una funzione di distribuzione sullo spazio delle fasi che dà per ogni macro configurazione la relativa probabilità. Nello studio macroscopico del sistema risulta invece più conveniente adottare una funzione di distribuzione ridotta, in cui alcuni gradi di libertà del sistema vengono tolti da un processo di media su tutte le possibili configurazioni. Così negli automi cellulari si usa la funzione di distribuzione a una particella $f_a(\vec{x}, t)$ che dà la probabilità di trovare una particella con velocità $\vec{e}_a$ nella posizione $\vec{x}$ al tempo t, mediata su tutte le possibili configurazioni dell'automa. La f_a può variare nel tempo per effetto di due processi: il movimento di particelle da un nodo all'altro (*propagazione*) e l'interazione di particelle all'interno di un singolo nodo (*collisione*). In assenza di collisioni la dinamica dell'automa si riassume nella seguente equazione [Wol86]:

$$f_a(\vec{x} + \vec{e}_a, t + 1) = f_a(\vec{x}, t) \tag{3.1}$$

Cioè una particella nella posizione $\vec{x}$ si muove al successivo istante di tempo nella cella adiacente di posizione $\vec{x}+\vec{e}_a$. Se consideriamo reticoli molto grandi ed intervalli di tempo lunghi, posizione e tempo possono essere approssimati da variabili continue: ciò permette quindi di espandere in serie di Taylor la (3.1) ottenendo la cosidetta *equazione di trasporto*:

$$\partial_t f_a(\vec{x},t) + \vec{e}_a \cdot \nabla f_a(\vec{x},t) = \Omega_a \tag{3.2}$$

Dove Ω_a rappresenta i cambiamenti della f_a dovuti a collisioni, che in generale dipenderanno non solo dalla distribuzione uniparticellare ma anche da funzioni di distribuzione a due e più particelle, dando luogo ad una dipendenza molto complicata. Nello sviluppo della teoria si ritiene però valida l'assunzione di *caos molecolare di Boltzmann*: ovvero Ω_a dipende solo dalle distribuzioni ad una particella. In particolare si assume che le particelle siano statisticamente non correlate prima di ogni collisione, in modo tale da scrivere le distribuzioni multiparticellari come prodotto delle distribuzioni uniparticellari. La (3.2) prende allora il nome di *equazione di trasporto di Boltzmann*. Per Ω_a valgono due *leggi di conservazione*:

$$\sum_a \Omega_a = 0 \tag{3.3}$$

$$\sum_a \vec{e}_a \Omega_a = \vec{0} \tag{3.4}$$

Queste due leggi non esprimono altro che la conservazione della massa (3.3) e della quantità di moto (3.4) in un processo di collisione.

Dalle quantità microscopiche si derivano le quantità macroscopiche *densità particellare* n:

$$\sum_a f_a = n$$

e *densità di momento* $n\vec{u}$, dove $\vec{u}$ è la velocità media del fluido:

$$\sum_a \vec{e}_a f_a = n\vec{u}$$

Se si ipotizza un equilibrio locale, ovvero probabilità invarianti nel tempo, allora $f_a(\vec{x},t)$ dipenderà solo dalle quantità macroscopiche $\vec{u}(\vec{x},t)$ e $n(\vec{x},t)$ e dalle loro derivate. Questa dipendenza può essere molto complicata, ma nei fenomeni idrodinamici u e n variano molto lentamente nel tempo e nello spazio (inoltre restando nel subsonico $|\vec{u}| \ll 1$), quindi con queste

assunzioni è possibile approssimare f_a con una espansione di Chapman-Enskog:

$$\begin{aligned} f_a =& f \left\{ 1 + c^{(1)} \vec{e_a} \cdot \vec{u} + c^{(2)} \left[(\vec{e_a} \cdot \vec{u})^2 - \frac{1}{d} |\vec{u}|^2 \right] + \right. \\ & \left. + c_\nabla^{(2)} \left[(\vec{e_a} \cdot \nabla)(\vec{e_a} \cdot \vec{u}) - \frac{1}{d} \nabla \cdot \vec{u} \right] + \cdots \right\} \end{aligned} \tag{3.5}$$

Dove i $c^{(i)}$ sono coefficienti da determinare, d è la dimensione dello spazio ($d = 3$ nel nostro caso) ed f è la probabilità all'equilibrio, cioè quando $\vec{u} = \vec{0}$. In generale $c^{(1)} = d$ mentre il valore esatto di $c^{(2)}$ e $c_\nabla^{(2)}$ richiederebbe la soluzione dell'equazione di trasporto (3.2), cosa in generale molto difficile a causa del termine di collisione. Comunque per la derivazione delle equazioni idrodinamiche non è necessario conoscere i valori esatti dei coefficienti: essi invece sono molto importanti per il calcolo dei termini viscosi. Per un automa cellulare il cui insieme dei velocity vectors è isotropo fino al quarto ordine si ha:

$$\nu = -\frac{c_\nabla^{(2)}}{d(d+2)} \tag{3.6}$$

I coefficienti viscosi per il modello rombododecaedro sono stati già calcolati [Sal94] e valgono:

- shear viscosity $\eta = -\frac{n}{12} c_\nabla^{(2)}$
- bulk viscosity $\eta = -\frac{n}{36} c_\nabla^{(2)}$

Quindi per il calcolo della viscosità è assolutamente necessaria la conoscenza di $c_\nabla^{(2)}$: verrà di seguito presentato un metodo per il calcolo approssimato di tale coefficiente, basato su un'approsimazione lineare all'equazione di trasporto di Boltzmann.

3.3 Approssimazione lineare

Nello studio del comportamento macroscopico di un sistema si può assumere che le distribuzioni di probabilità f_a differiscano poco dai loro valori di equilibrio, come è supposto nell'espansione di Chapman-Enskog (3.5). Quindi f_a può essere approssimata nel seguente modo:

$$f_a = f(1 - \phi_a) \qquad (|\phi_a| \ll 1). \tag{3.7}$$

Con questa approssimazione e tenendo presente l'ipotesi di caos di Boltzmann il termine di collisione Ω_a della (3.2) può essere a sua volta approssimato da una espansione in serie di potenze, che per i nostri scopi è sufficente considerare fino al termine lineare.

Supponiamo Ω_a funzione del vettore Φ di componenti $\phi_1, \ldots, \phi_r$, dove r è la cardinalità dell'insieme dei velocity vectors. Una espansione in serie di potenze centrata in $\vec{0}$ dà:

$$\Omega_a(\Phi) = \Omega_a(\vec{0}) + (\nabla_{\vec{0}}\Omega_a) \cdot \Phi + \cdots \tag{3.8}$$

Consideriamo il termine costante $\Omega_a(\vec{0})$: la (3.4) implica che per $\phi_a = 0$ allora $f_a = f$, ovvero il fluido è in condizioni di equilibrio ($\vec{u} = 0$). Ma in queste condizioni il contributo delle collisioni è nullo, perciò $\Omega_a(\vec{0}) = 0$. L'espansione (3.4) può essere quindi riscritta come:

$$\Omega_a(\Phi) = \sum_{b=1}^{r} \omega_{ab}\, \phi_b \tag{3.9}$$

in cui i coefficienti ω_{ab} valgono:

$$\omega_{ab} = \frac{\partial \Omega_a(\vec{0})}{\partial \phi_b} \tag{3.10}$$

Applicando le leggi di conservazione (3.3) e (3.4) alla (3.9) possono essere derivate due condizioni per i coefficenti ω_{ab}. Per la conservazione della massa nelle collisioni si ha che:

$$\sum_b \omega_{ab} = 0 \tag{3.11}$$

Mentre per la conservazione della quantità di moto vale:

$$\sum_b \vec{e}_a \omega_{ab} = \vec{0} \tag{3.12}$$

3.4 Valutazione dei coefficienti

L'espansione di Chapman-Enskog (3.5) dà la forma generale per le approssimazioni alla distribuzione f_a. Il coefficiente $c_\nabla^{(2)}$ che compare in tale espansione può essere stimato confrontando l'espansione di Chapman-Enskog con l'equazione di trasporto di Boltzmann in cui è considerata l'ipotesi di *equilibrio microscopico*:

$$\partial_t f_a = 0 \tag{3.13}$$

Quindi l'equazione di Boltzmann (3.2) diventa:

$$\vec{e}_a \cdot \nabla f_a(\vec{x}, t) = \Omega_a \tag{3.14}$$

Applicando la (3.4) possiamo ulteriormente trasformare la (3.14) in:

$$\vec{e}_a \cdot \nabla(f(1 - \phi_a)) = \Omega_a \tag{3.15}$$

A questo punto possiamo sostituire Ω_a con l'approssimazione lineare (3.7) e svolgere alcuni passaggi algebrici:

$$-f\vec{e}_a \cdot \nabla\phi_a = \sum_{b=1}^{r} \omega_{ab}\,\phi_b \tag{3.16}$$

Tenendo conto dell'ipotesi di incopressibilità $\nabla \cdot \vec{u} = 0$, l'espansione di Chapman-Enskog (3.5) senza i termini quadratici fornisce la forma per il coefficiente ϕ_b:

$$\phi_b = -c^{(1)}(\vec{e}_b \cdot \vec{u}) - c_\nabla^{(2)}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) \tag{3.17}$$

Sostituiamo la forma appena trovata nella (3.16) e svolgiamo alcuni passaggi algebrici:

$$-f\vec{e}_a \cdot \nabla\left[-c^{(1)}(\vec{e}_a \cdot \vec{u}) - c_\nabla^{(2)}(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})\right] = -\sum_{b=1}^{r} \omega_{ab}\left[c^{(1)}(\vec{e}_b \cdot \vec{u}) + c_\nabla^{(2)}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u})\right]$$

$$c^{(1)} f\vec{e}_a \cdot \nabla(\vec{e}_a \cdot \vec{u}) + c_\nabla^{(2)} f\vec{e}_a \cdot \nabla\left[(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})\right] = -c^{(1)} \sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \vec{u}) + -c_\nabla^{(2)} \sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u})$$

$$c^{(1)} f(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u}) + c_\nabla^{(2)} f\vec{e}_a \cdot \nabla\left[(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})\right] = -c^{(1)} \sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \vec{u}) + -c_\nabla^{(2)} \sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u})$$

Escludendo i termini differenziali di ordine due si ottiene:

$$\begin{aligned} c^{(1)} f(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u}) = -c^{(1)} &\sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \vec{u}) + \\ -c_\nabla^{(2)} &\sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) \end{aligned} \tag{3.18}$$

Sviluppiamo la prima sommatoria della precedente equazione ricordando la legge di conservazione (3.12):

$$\sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \vec{u}) = \sum_{b=1}^{r} (\omega_{ab}\vec{e}_b \cdot \vec{u}) = \left(\sum_{b=1}^{r} \omega_{ab}\vec{e}_b \right) \cdot \vec{u} = \vec{0} \cdot \vec{u} = 0$$

Quindi la (3.18) diventa:

$$c^{(1)} f(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u}) = -c_{\nabla}^{(2)} \sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) \tag{3.19}$$

Da questa relazione si ricava subito il coefficiente $c_{\nabla}^{(2)}$ cercato:

$$c_{\nabla}^{(2)} = -\frac{c^{(1)} f(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})}{\sum_{b=1}^{r} \omega_{ab}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u})} \tag{3.20}$$

Quindi una volta noti i coefficienti ω_{ab} tramite la (3.20) è possibile dare un valore approssimato della viscosità.

3.5 Esempio applicativo

Consideriamo un automa cellulare in cui l'insieme V dei velocity vectors sia formato da M vettori e_a appartenenti ad uno spazio $r-$dimensionale. Supponiamo che tale insieme sia invariante rispetto al ribaltamento di coordinate: cioè se $e_a \in V$ allora $-e_a \in V$. In un tale automa due particelle con velocità e_a e $-e_a$ possono perciò collidire e distribuirsi in due direzioni e_b e $-e_b$ con $b \neq a$. Le direzioni possibili sono chiaramente più di una, esattamente $M-2$, quindi si pone il problema di quale configurazione post-collisione scegliere: una semplice soluzione è quella di stabilire una regola probabilistica che assegni ad ogni configurazione post-collisione ammissibile la stessa probabilità $p = \frac{2}{M-2}$. Tale tipo di collisioni viene detto *head on*.

Per il calcolo dei coefficienti ω_{ab} dobbiamo adesso scrivere il termine Ω_a dell'equazione di Boltzmann (3.2) per questo automa di esempio.

Il processo di collisione si può formalizzare in una regola di transizione non deterministica che cambia uno stato pre-collisione $s = \{s_i,\, i = 1, \ldots, r\}$ in uno stato $s' = \{s'_i,\, i = 1, \ldots, r\}$. Definiamo:

- $p_{s,s'}$ probabilità di transizione dallo stato s allo stato s';
- $P(s)$ la probabilità dello stato s.

Con queste definizioni il termine di collisione è:

$$\Omega_a = \sum_{s,s\prime}(s'_a - s_a)p_{s,s\prime}P(s) \tag{3.21}$$

Dividiamo la sommatoria in due parti:

$$\sum_{s|s_a=1,s'}(s'_a - s_a)p_{s,s'}P(s) + \sum_{s|s_a=0,s'}(s'_a - s_a)p_{s,s'}P(s)$$

Dalla prima sommatoria si possono escludere i termini in cui $s'_a = 1$ perchè danno contributo nullo e la stessa cosa si può fare nella seconda sommatoria per i termini in cui $s'_a = 0$. Quindi:

$$\begin{aligned}\Omega_a = &\sum_{\substack{s|s_a=1\\ s'|s'_a=0}}(s'_a - s_a)p_{s,s'}P(s) + \sum_{\substack{s|s_a=0\\ s'|s'_a=1}}(s'_a - s_a)p_{s,s'}P(s) = \\ = &\sum_{\substack{s|s_a=1\\ s'|s'_a=0}}(-1)p_{s,s'}P(s) + \sum_{\substack{s|s_a=0\\ s'|s'_a=1}}p_{s,s'}P(s)\end{aligned}$$

Ma $p_{s,s'} = \frac{2}{M-2}$ perciò la precedente diventa:

$$\Omega_a = -\sum_{s|s_a=1}\frac{2}{M-2}P(s) + \sum_{s|s_a=0}\frac{2}{M-2}P(s) \tag{3.22}$$

La probabilità di uno stato s è, per l'ipotesi di Boltzmann [FdH$^+$88]:

$$P(s) = \prod_i f_i^{s_i}(1-f_i)^{(1-s_i)} \tag{3.23}$$

La prima sommatoria della (3.22) rappresenta la somma delle probabilità degli stati di collisione in cui $s_a = 1$ e $s_{\bar{a}} = 1$ (quindi nelle altre direzioni non ci sono particelle), dove con $\bar{a}$ si intende la direzione opposta ad a. Quindi:

$$\Omega_a = -\frac{2}{M-2}\sum_{s|s_a=1} f_a f_{\bar{a}} \prod_{b\neq a,\bar{a}}(1-f_b) + \sum_{s|s_a=0}\frac{2}{M-2}P(s)$$

Ma gli stati che soddisfano la condizione sono $(M-2)/2$ quindi:

$$\Omega_a = -\frac{2}{M-2}\frac{M-2}{2} f_a f_{\bar{a}} \prod_{b\neq a,\bar{a}}(1-f_b) + \sum_{s|s_a=0}\frac{2}{M-2}P(s)$$

La seconda sommatoria della (3.22) rappresenta invece la somma delle probabilità degli stati di collisione in cui sia s_a sia $s_{\bar{a}}$ valgono 0 mentre nelle altre direzioni opposte prese due a due ci sono particelle. Abbiamo quindi:

$$\begin{aligned}\Omega_a &= -f_a f_{\bar{a}} \prod_{b\neq a,\bar{a}} (1-f_b) + \frac{1}{2}\frac{2}{M-2} \sum_{b\neq a,\bar{a}} f_b f_{\bar{b}} \prod_{c\neq b,\bar{b}} (1-f_c) = \\ &= -f_a f_{\bar{a}} \prod_{b\neq a,\bar{a}} (1-f_b) + \frac{1}{M-2} \sum_{b\neq a,\bar{a}} f_b f_{\bar{b}} \prod_{c\neq b,\bar{b}} (1-f_c)\end{aligned} \tag{3.24}$$

A questo punto applichiamo l'approssimazione (3.7) alla precedente forma per Ω_a e svolgiamo alcuni passaggi algebrici:

$$\begin{aligned}\Omega_a &= -f(1-\phi_a) f(1-\phi_{\bar{a}}) \prod_{b\neq a,\bar{a}} (1-f(1-\phi_b)) + \\ &+ \frac{1}{M-2} \sum_{b\neq a,\bar{a}} f^2(1-\phi_b)(1-\phi_{\bar{b}}) \prod_{c\neq b,\bar{b}} (1-f(1-\phi_c)) = \\ &= -f^2(1-\phi_a-\phi_{\bar{a}}+\phi_a\phi_{\bar{a}}) \prod_{b\neq a,\bar{a}} [(1-f)+f\phi_b] + \\ &+ \frac{1}{M-2} \sum_{b\neq a,\bar{a}} f^2(1-\phi_b-\phi_{\bar{b}}+\phi_b\phi_{\bar{b}}) \prod_{c\neq b,\bar{b}} [(1-f)+f\phi_c] =\end{aligned}$$

In forza all'approssimazione lineare (3.9) dobbiamo scartare i termini quadratici:

$$\begin{aligned}\Omega_a &= -f^2(1-\phi_a-\phi_{\bar{a}}) \prod_{b\neq a,\bar{a}} [(1-f)+f\phi_b] + \\ &+ \frac{1}{M-2} \sum_{b\neq a,\bar{a}} f^2(1-\phi_b-\phi_{\bar{b}}) \prod_{c\neq b,\bar{b}} [(1-f)+f\phi_c] =\end{aligned} \tag{3.25}$$

Sviluppiamo il primo addendo della (3.25):

$$\begin{aligned}\Omega_a &= (-f^2+f^2(\phi_a+\phi_{\bar{a}})) \left[(1-f)^{M-2} + \sum_{b\neq a,\bar{a}} (1-f)^{M-3} f\phi_b\right] + \\ &+ \frac{1}{M-2} \sum_{b\neq a,\bar{a}} f^2(1-\phi_b-\phi_{\bar{b}}) \prod_{c\neq b,\bar{b}} [(1-f)+f\phi_c]\end{aligned}$$

Escludendo termini quadratici e costanti si ha:

$$\begin{aligned}\Omega_a = &-f^3(1-f)^{M-3}\sum_{b\neq a,\bar{a}}\phi_b + f^2(1-f)^{M-2}(\phi_a+\phi_{\bar{a}})+\\ &+\frac{1}{M-2}\sum_{b\neq a,\bar{a}} f^2(1-\phi_b-\phi_{\bar{b}})\prod_{c\neq b,\bar{b}}[(1-f)+f\phi_c]\end{aligned} \tag{3.26}$$

Sviluppiamo ora la seconda sommatoria nella (3.26) applicando le stesse considerazioni precedenti:

$$\begin{aligned}\Omega_a &= -f^3(1-f)^{M-3}\sum_{b\neq a,\bar{a}}\phi_b + f^2(1-f)^{M-2}(\phi_a+\phi_{\bar{a}})+\\ &+\frac{1}{M-2}\sum_{b\neq a,\bar{a}}(f^2-f^2(\phi_b-\phi_{\bar{b}}))\left[(1-f)^{M-2}+\sum_{c\neq b,\bar{b}} f(1-f)^{M-3}\phi_c\right] =\\ &= -f^3(1-f)^{M-3}\sum_{b\neq a,\bar{a}}\phi_b + f^2(1-f)^{M-2}(\phi_a+\phi_{\bar{a}})+\\ &+\frac{1}{M-2}\sum_{b\neq a,\bar{a}}\left[-f^2(1-f)^{M-2}(\phi_b-\phi_{\bar{b}})+f^3(1-f)^{M-3}\sum_{c\neq b,\bar{b}}\phi_c\right] =\\ &= -f^3(1-f)^{M-3}\sum_{b\neq a,\bar{a}}\phi_b + f^2(1-f)^{M-2}(\phi_a+\phi_{\bar{a}})+\\ &+\frac{f^3(1-f)^{M-3}}{M-2}\sum_{b\neq a,\bar{a}}\sum_{c\neq b,\bar{b}}\phi_c - \frac{f^2(1-f)^{M-2}}{M-2}\sum_{b\neq a,\bar{a}}(\phi_b+\phi_{\bar{b}}) =\\ &= -f^3(1-f)^{M-3}\sum_{b\neq a,\bar{a}}\phi_b + f^2(1-f)^{M-2}(\phi_a+\phi_{\bar{a}})+\\ &+f^3(1-f)^{M-3}\sum_{b}\phi_b - \frac{f^3(1-f)^{M-3}}{M-2}\sum_{b\neq a,\bar{a}}\phi_b+\\ &-\frac{f^2(1-f)^{M-2}}{M-2}\sum_{b\neq a,\bar{a}}(\phi_b+\phi_{\bar{b}}) =\\ &= f^3(1-f)^{M-3}(\phi_a+\phi_{\bar{a}}) + f^2(1-f)^{M-2}(\phi_a+\phi_{\bar{a}})+\\ &-\frac{2}{M-2}f^3(1-f)^{M-3}\sum_{b\neq a,\bar{a}}\phi_b - \frac{2}{M-2}f^2(1-f)^{M-2}\sum_{b\neq a,\bar{a}}\phi_b\end{aligned}$$

Dopo altri passaggi algebrici otteniamo la forma finale del termine di collisione:

$$\Omega_a = f^2(1-f)^{M-3}\left[(\phi_a + \phi_{\bar{a}}) - \frac{1}{M-2}\sum_{b\neq a,\bar{a}}(\phi_b + \phi_{\bar{b}})\right] \tag{3.27}$$

Da quest'ultima relazione ricaviamo direttamente i coefficienti ω_{ab} necessari:

$$\omega_{ab} = \begin{cases} f^2(1-f)^{M-3} & b = a, \bar{a} \\ \frac{2}{M-2} f^2(1-f)^{M-3} & b \neq a, \bar{a} \end{cases}$$

Ora non ci resta che sostituire questi coefficienti nella (3.20) per ottenere il valore del coefficiente $c_\nabla^{(2)}$:

$$c_\nabla^{(2)} = -\frac{c^{(1)} f(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})}{\sum_b \omega_{ab}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u})}$$

Consideriamo solo il denominatore:

$$\begin{aligned}
\sum_b \omega_{ab}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) &= 2f^2(1-f)^{M-3}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) + \\
&+ \frac{2}{M-2} f^2(1-f)^{M-3} \sum_{b\neq a,\bar{a}} (\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) = \\
&= 2f^2(1-f)^{M-3}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) + \\
&- \frac{2}{M-2} f^2(1-f)^{M-3}\left[\sum_b (\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) + \right. \\
&\left. -2(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})\right] = \\
&= 2f^2(1-f)^{M-3}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) + \\
&- \frac{2}{M-2} f^2(1-f)^{M-3}\left[\frac{M}{d}(\nabla \cdot \vec{u}) + \right. \\
&\left. -2(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})\right]
\end{aligned}$$

Applicando la condizione di incompressibilità $\nabla \cdot \vec{u} = 0$:

$$\sum_b \omega_{ab}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) = 2f^2(1-f)^{M-3}(\vec{e}_b \cdot \nabla)(\vec{e}_b \cdot \vec{u}) + \\ + \frac{4}{M-2} f^2(1-f)^{M-3} \left[(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})\right] = \\ = \frac{2M}{M-2} f^2(1-f)^{M-3}(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u})$$

Quindi la forma per $c_\nabla^{(2)}$ è:

$$c_\nabla^{(2)} = -\frac{(M-2)c^{(1)}f}{2Mf^2(1-f)^{M-3}}$$

Sostituendo $c_\nabla^{(2)}$ nella (3.6) e ricordando che $c^{(1)} = d$ si ha una forma della viscosità per un generico automa cellulare con collisioni head-on e velocity set isotropo fino al quarto ordine:

$$\nu = \frac{(M-2)}{2M(d+2)f(1-f)^{M-3}} \tag{3.28}$$

Esempio 3.5.1. Consideriamo il modello di automa cellulare bidimensionale FHP. Per tale modello si ha $M = 6$ e $d = 2$ quindi la viscosità è:

$$\nu = \frac{1}{12f(1-f)^3}$$

Se invece si considerano collisioni non solo a due particelle la viscosità è stata dimostrata valere:

$$\nu = \frac{1}{12f(1-f)^3} - \frac{1}{8}$$

Quindi l'aumento degli stati di collisione ha portato solo una diminuzione costante della viscosità: da ciò si deduce che le collisioni in un certo senso più importanti sono quelle head-on a due particelle.

Esempio 3.5.2. Consideriamo un invece un velocity set composto dai vertici di un icosaedro: si ha $M = 12$, $d = 3$ e l'isotropia del tensore di quarto ordine. La viscosità è quindi:

$$\nu = \frac{1}{12f(1-f)^9}$$

Purtroppo l'icosaedro non riempe perfettamente lo spazio tridimensionale e quindi non può essere usato come modello di automa cellulare.

3.6 Coefficienti del modello rombododecaedrico

Applichiamo qui il metodo sviluppato nella sezione (3.4) al nostro automa cellulare. Adottando la stessa notazione della sezione (3.5) il termine di collisione Ω_a è:

$$\Omega_a = \sum_{s,s'} (s_a - s'_a)\, p_{s,s'} P(s) =$$

$$= \sum_{s,s'} (s_a - s'_a)\, p_{s,s'} \prod_b f_b^{s_b} (1-f_b)^{1-s_b} =$$

$$= \sum_{\substack{s|s_a=0 \\ s'|s'_a=1}} (-1)\, p_{s,s'} \prod_b f_b^{s_b} (1-f_b)^{1-s_b} +$$

$$+ \sum_{\substack{s|s_a=1 \\ s'|s'_a=0}} (1)\, p_{s,s'} \prod_b f_b^{s_b} (1-f_b)^{1-s_b} =$$

$$= \sum_{\substack{s|s_a=0 \\ s'|s'_a=1}} (-1)\, p_{s,s'} (1-f_a) f^{|s|} \prod_{b\neq a} (1-f_b)^{1-s_b} +$$

$$+ \sum_{\substack{s|s_a=1 \\ s'|s'_a=0}} (1)\, f_a\, f^{|s|}\, p_{s,s'} \prod_{b\neq a} (1-f_b)^{1-s_b}$$

Dove $|s|$ rappresenta il numero di particelle per uno stato s. Per la (3.7):

$$\Omega_a = - \sum_{\substack{s|s_a=0 \\ s'|s'_a=1}} p_{s,s'} f^{|s|} \underbrace{(1-f+f\phi_a)\prod_{b\neq a}(1-\phi_b)^{s_b}(1-f+f\phi_b)^{1-s_b}}_{A} +$$

$$+ \sum_{\substack{s|s_a=1 \\ s'|s'_a=0}} p_{s,s'} f^{|s|} \underbrace{f(1-\phi_a)\prod_{b\neq a}(1-\phi_a)^{s_b}(1-f+\phi_b)^{1-s_b}}_{B} \tag{3.29}$$

Sviluppiamo solo l'argomento della prima sommatoria della precedente equazione considerando stati di collisione per cui $s_a = 0$ ed escludendo eventuali termini quadratici nelle variabili ϕ_i :

$$A = [(1-f)+f\phi_a)]\prod_{b\neq a}(1-\phi_b)^{s_b}\,[(1-f)+f\phi_b]^{1-s_b} =$$

$$= \underbrace{(1-f+f\phi_a)(1-\phi_b)(1-\phi_c)\prod_{d\neq a,b,c}(1-f+f\phi_d)}_{\text{stati a 2 particelle}} =$$

$$= \underbrace{(1-f+f\phi_a)(1-\phi_b)(1-\phi_c)(1-\phi_d)\prod_{e\neq a,b,c,d}(1-f+f\phi_e)}_{\text{stati a 3 particelle}} = \cdots$$

$$\cdots = \underbrace{f(1-f)^{M-11}(\phi_a+\phi_n) - \sum_{r=b}^{m}(1-f)^{M-10}\phi_r}_{\text{stati a 10 particelle}}$$

Si considerano solo stati con numero di particelle compreso tra due e dieci perchè il modello non ammette collisioni a una e undici particelle. Riscri-

viamo la (3.29) tenendo conto dell'approssimazione (3.9):

$$\Omega_a = - \sum_{\substack{s|s_a=0 \\ s'|s'_a=1}} p_{s,s'} f^{|s|} \left[f(1-f)^{M-|s|-1}(\phi_a + \phi_n) - (1-f)^{M-|s|} \sum_{r=b}^{m} \phi_r \right] +$$
$$+ \sum_{\substack{s|s_a=1 \\ s'|s'_a=0}} p_{s,s'} f^{|s|} \underbrace{f(1-\phi_a) \prod_{b \neq a} (1-\phi_a)^{s_b} (1-f+\phi_b)^{1-s_b}}_{B}$$

Sviluppiamo adesso l'argomento della seconda sommatoria della precedente equazione considerando stati di collisione per cui $s_a = 1$ ed escludendo eventuali termini quadratici nelle variabili ϕ_i :

$$B = \underbrace{f(1-\phi_a)(1-\phi_b) \prod_{c \neq a,b} (1-f+f\phi_c)}_{\text{stati a 2 particelle}} =$$
$$= \underbrace{f(1-\phi_a)(1-\phi_b)(1-\phi_c) \prod_{d \neq a,b,c} (1-f+f\phi_d)}_{\text{stati a 3 particelle}} = \cdots$$
$$\cdots = \underbrace{f \left[f(1-f)^{M-11}(\phi_m + \phi_n) - (1-f)^{M-10} \sum_{r=a}^{l} \phi_r \right]}_{\text{stati a 10 particelle}}$$

Quindi la (3.29) risulta:

$$\Omega_a = - \sum_{\substack{s|s_a=0 \\ s'|s'_a=1}} p_{s,s'} f^{|s|} \left[f(1-f)^{M-|s|-1}(\phi_a + \phi_n) - (1-f)^{M-|s|} \sum_{r=b}^{m} \phi_r \right] +$$
$$- \sum_{\substack{s|s_a=1 \\ s'|s'_a=0}} p_{s,s'} f^{|s|+1} \left[(1-f)^{M-|s|}(\phi_n + \phi_m) - (1-f)^{M-|s|} \sum_{r=a}^{l} \phi_r \right] \tag{3.30}$$

Così come è la precedente equazione non consente di ricavare i coefficienti ω_{ab} cercati: bisogna prima effettuare una trasformazione. Consideriamo

le sommatorie non su tutto l'insieme degli stati di collisione ma su una sua partizione basata sul numero di particelle per stato. Applicando tale ragionamento alla (3.30) dopo alcuni passaggi algebrici si arriva alla:

$$\Omega_a = \sum_{i=2}^{12} \phi_i \left[\sum_{\substack{s|s_a=0 \\ s'|s'_a=1 \\ 2\leq|s|\leq i-2}} p_{s,s'} f^{|s|+1}(1-f)^{M-|s|-1} + \right.$$

$$- \sum_{\substack{s|s_a=0 \\ s'|s'_a=1 \\ |s|\geq i-1}} p_{s,s'} f^{|s|}(1-f)^{M-|s|} + \sum_{\substack{s|s_a=1 \\ s'|s'_a=0 \\ 2\leq|s|\leq i-1}} p_{s,s'} f^{|s|+2}(1-f)^{M-|s|-1} +$$

$$\left. - \sum_{\substack{s|s_a=1 \\ s'|s'_a=0 \\ |s|\geq i}} p_{s,s'} f^{|s|+1}(1-f)^{M-|s|} \right] \tag{3.31}$$

Sebbene la precedente permetta di ricavare i coefficienti delle variabili ϕ_i è ancora da stabilire se questi coefficienti dipendano o meno dalla particolare direzione a. Quindi la (3.31) necessita di ulteriori approfondimenti per eliminare la dipendenza dei coefficienti dalla particolare direzione. Una volta fatto questo si potrà usare la (3.20) per il calcolo del coefficiente viscoso $c_\nabla^{(2)}$.

Capitolo 4

Calcolo parallelo e Cray

4.1 Introduzione

In questo capitolo verrà illustrata l'architettura hardware/software dei sistemi a parallelismo massiccio prodotti dalla Cray Research. In particolare verranno discussi i paradigmi di programmazione disponibili e loro effettiva usabilità nell'implementazione del software di simulazione.

4.2 Architettura hardware

I sistemi Cray T3D/E fanno parte della famiglia MPP (Massively Parallel Processing) e sono sistemi di calcolo massicciamente paralleli costruiti sulla base di microprocessori commerciali (Alpha Digital). La loro architettura è stata progettata per gestire al massimo 2048 processori. Tale macchine sono classificabili come sistemi MIMD (Multiple Instruction Multiple Data) con in più la possibilità di simulare sistemi SIMD (Single Instruction Multiple Data) e SPMD (Same Program Multiple Data). La topologia di interconnessione tra i processori è toroidale in tre dimensioni.

La memoria è fisicamente distribuita ma condivisa logicamente: ogni processore ha la proprio memoria ma può indirizzare la memoria di un qualsiasi altro processore. Questo tipo di accesso viene detto NUMA (Non Uniform Memory Access) perchè l'accesso a memoria si svolge con velocità diverse a seconda della effettiva dislocazione (remota o locale) della parola indirizzata. Entrambi i sistemi dispongono di dispositivi hardware per il *latency hiding* cioè il mascheramento dei tempi di latenza nella comunicazione tra processori.

4.2.1 Cray T3D

Il Processing Element (PE) del Cray T3D è il microprocessore Digital Alpha EV4 con frequenza di clock a 150 MHz. Questo microprocessore è un RISC interamente a 64 bit con cache memory interna di 16Kbyte e raggiunge la velocità di 150 Mflop/s. La cache è di tipo direct-mapped ed è organizzata in 256 linee di 32 bytes ciascuna. I programmi possono invalidare la cache locale se questo è necessario per mantenere la coerenza con la memoria principale. Tra il processore e la memoria non c'è alcuna cache di secondo livello: una mancanza che si nota sopratutto nei programmi con scarsa località negli accessi a memoria.

I meccanismi di latency hiding del T3D sono:

- una *Pre-Fetch Queue (PFQ)*;
- il *Block Transfer Engine (BTE)*;

La Pre-Fetch Queue è una coda FIFO di 16 elementi e consente di aggirare il ritardo del microprocessore nelle operazioni di singola lettura in memoria remota. Il processore può emettere particolare istruzioni prefetch e successivamente andare ai dati nella PFQ. Data la dimensione della coda tale sistema è utile solo in caso di singole e sporadiche letture remote.

Il Block Transfer Engine è un dispositivo asincrono di accesso alla memoria : può trasferire pacchetti fino a 64Kbyte di dati senza interrompere l'attività del PE e funziona in maniera bidirezionale tra memoria locale e memoria remota. Questo dispositivo è utile per grossi trasferimenti asincroni di dati in quanto ha un certo costo di avvio.

Le rete di interconnessione collega quelli che vengono chiamati *nodi computazionali*. Un nodo computazionale è formato da:

- due PE con la relativa memoria (massimo 64 Mbyte per PE);
- una interfaccia per la rete di interconnesione;
- un Block Transfer Engine;

La figura 4.1 mostra la struttura di un nodo.

La rete lavora indipendentemente dai processori, quindi i dati viaggiano da un nodo all'altro senza interrompere sia i processori lungo il percorso sia il processore di destinazione. La larghezza di banda è di circa 300 Mbyte/s mentre la velocitè reale è di circa 130 Mbyte/s per letture remote e di circa 160 Mbyte/s per scritture remote. Quindi nel caso di scambio di dati tra processori è conveniente fare delle scritture remote anche se questo, come

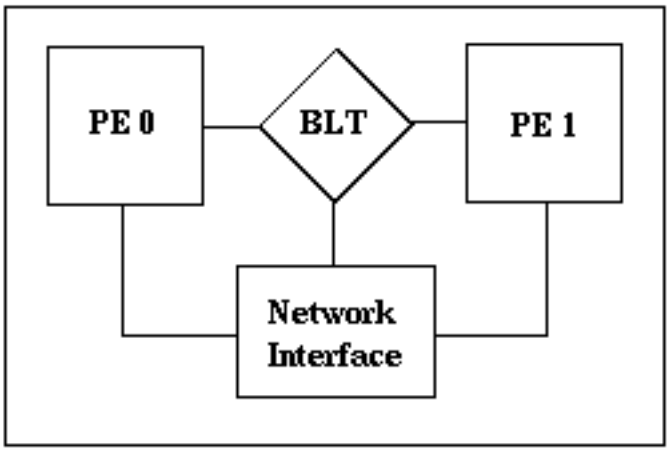

Figura 4.1: struttura di un nodo T3D

vedremo, porta ad una complicazione della parte di sincronizzazione. L'algoritmo di instradamento usato è un *dimension order routing* di tipo statico: i dati viaggiano prima nella direzione X, poi nella direzione Y ed infine nella direzione Z. Il sistema operativo definisce delle tabelle di instradamento per ogni nodo con il percorso ottimizzato tra i nodi.

Parallela alla rete di comunicazione, tra i nodi esiste un rete dedicata solamente alla sincronizzazione tra processori. Può funzionare in due modi:

barrier mode : segnala quando l'ultimo PE raggiunge un determinato punto all'interno del codice;

eureka mode : segnala quando il primo PE raggiunge un determinato punto del codice.

Il T3D non è un sistema autonomo: ha bisogno di un calcolatore host che gli fornisca tutte le funzionalità di base di sistema operativo. Nella pratica ciò si è visto costituire un notevole collo di bottiglia, a causa della frequente comunicazione tra host e T3D.

4.2.2 Cray T3E

La macro architettura del Cray T3E è la stessa del T3D: un sistema MIMD di tipo NUMA. La rete di interconnesione è sempre a topologia toroidale in tre dimensioni, ma sono potenziati i meccanismi di sincronizzazione e di latency hiding. La larghezza di banda è ora di 480 Mbyte/s in qualsiasi

direzione, mentre la latenza hardware è inferiore a $1\mu s$. Oltre al dimension order routing di tipo statico il T3E supporta un *routing adattivo* dinamico.

La rete di sincronizzazione è stata eliminata, in quanto il potenziamento della rete di interconnesione consente di inviare sia le informazioni sia gestire la sincronizzazione sulla stessa rete. Inoltre ogni PE è direttamente connesso alla rete.

Il meccanismo principale di latency hiding è costituito dagli *E-registers*, 512 registri memory mapped che permettono letture e scritture remote. I dati possono essere caricati dagli E-register ai registri del processore alla velocità di 600 Mbyte/s. Questo dispositivo sostituisce la Pre Fetch Queue e il Block Transfer Engine del T3D e a differenza del T3D permette la stessa velocità di 380 Mbyte/s sia in lettura remota sia scrittura remota.

Il PE è il microprocessore Digital Alpha EV5 con frequenza di clock di 300 MHz, per una velocità di picco di 600 Mflop/s. Oltre alla cache interna è presente una cache secondaria di 96Kbyte e degli *stream buffers* che consentono il precaricamento dei dati dalla memoria a buffers veloci, il tutto gestito automaticamente dall'hardware. L'hardware gestisce inoltre la coerenza della cache con la memoria centrale. Ogni processore può disporre al massimo di 512 Mbyte di memoria locale.

Il singolo nodo del T3E, come si può vedere dalla figura 4.2 è costituito da quattro elementi:

- processore;
- logica di controllo;
- memoria;
- router;

I nodi sono raggruppati a gruppi di quattro e formano un *modulo*. Ogni modulo è connesso alla Giga Ring Interface per la gestione dell'I/O. Ogni Giga Ring Interface è a sua volta connessa a due canali a 32 bit collegati ad anello (Giga Ring), che presentano una larghezza di banda di 1.2Gbyte/s; tali canali possono connettere più sistemi e in particolare tutte le periferiche di I/O.

Il T3E a differenza del T3D è un sistema completamente autonomo, cioè non ha più bisogno di una macchina front-end che gli fornisca le funzionalità di base di sistema operativo. Infatti la macchina è dotata di un sistema operativo distribuito tipo Unix basato su tecnologia a microkernel.

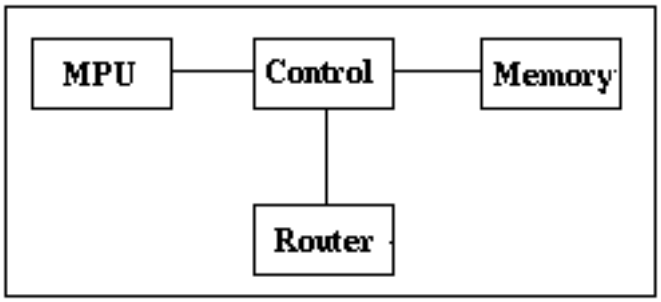

Figura 4.2: struttura di un nodo T3E

Tale sistema operativo distingue fra diversi tipi di PE, a seconda dell'uso cui sono dedicati:

command PEs : usati solo dal sistema operativo;

command PEs : dedicati all'esecuzione di comandi;

application PEs : dedicati alle applicazioni multiprocessore.

4.3 Paradigmi di programmazione

I sistemi Cray MPP supportano quattro distinti, ma cooperanti, paradigmi di programmazione: *data sharing, work sharing, message passing e shared memory (SHMEM)*. Questi metodi non sono mutuamente esclusivi; parciò possono essere mescolati a piacere nello sviluppo di una applicazione.

I paradigmi si possono dividere in due classi discriminando sulla modalità di comunicazione adottata: *implicita o esplicita*.

I paradigmi con comunicazione implicita sono:

data sharing : disponibile solo in Fortran, distribuisce i dati (ad esempio un array) nelle memorie dei PE che stanno elaborando l'applicazione;

work sharing : disponibile solo in Fortran, distribuisce gli statements del programma tra i PE, con l'obiettivo di eseguirli in parallelo. Ad esempio è possibile dividere le iterazione di un ciclo `DO` tra i processori.

I paradigmi con comunicazione esplicita sono:

message passing : è il noto scambio di messaggi ed è disponibile in Fortran, C e C++. È implementato secondo due protocolli: PVM (Parallel Virtual Machine) e MPI (Message Passing Interface).

shared memory : disponibile in Fortran, C e C++ è il paradigma di programmazione più esplicito ed è composto da routine di basso livello che operano una comunicazione senza sincronizzazione.

Nella scelta del metodo da usare nell'implementazione del software di simulazione sono stati subito scartati i metodi a comunicazione implicita, in quanto disponibili solo in Fortran. Le specifiche del software richiedevano invece la codifica in linguaggio ANSI C. Tra i paradigmi a comunicazione esplicita è stato scelto lo shared memory (SHMEM) in quanto più efficiente. La maggiore efficienza deriva dal fatto che l'SHMEM è un paradigma a basso livello che rispecchia la struttura dell'hardware sottostante. Nel seguito analizzeremo quindi solo tale metodo.

4.3.1 Paradigma shared memory (SHMEM)

SHMEM permette una comunicazione strutturata attraverso la memoria condivisa nei sistemi Cray MPP. Si ricorda che nei sistemi Cray MPP la memoria è fisicamente distribuita ma logicamente condivisa, cioè un processore può indirizzare una qualsiasi parola della memoria di un qualsiasi processore remoto. Tale metodo è il più esplicito tra quelli disponibili e il programmatore è direttamente responsabile della sincronizzazione, della coerenza della cache e degli effettivi trasferimenti dei dati. SHMEM è implementata sotto forma di libreria di funzioni da collegare all'applicazione in fase di linking.

La comunicazione avviene specificando *due indirizzi*: un *indirizzo locale* e un *indirizzo remoto* nella forma (numero PE, indirizzo locale).

Informazioni di sistema

La libreria fornisce alcune funzioni per ottenere informazioni sulla configurazione di sistema. Le due principali sono: `_num_pes()` e `_my_pe()`.

La funzione `_num_pes()` ritorna il numero di PE disponibili per l'applicazione; questa funzione è valutata a tempo di esecuzione e quindi non può essere usata in espressioni costanti. La dichiarazione è :

```
int _num_pes (void)
```

Questa funzione potrebbe per esempio essere usata per allocare un array di lunghezza variabile la cui dimensione è data dal numero di processori disponibili:

```
funzione_di_esempio()
 {
   int somma[_num_pes()];
   .....
 }
```

La funzione `_my_pe()` ritorna il numero del processore sul quale è in esecuzione:

```
int _my_pe (void)
```

Il valore di ritorno è un intero compreso tra 0 e `_num_pes()`-1. Un uso frequente di questa funzione è mostrato nel seguente esempio, in cui viene diviso il lavoro tra i vari PE:

```
switch (_my_pe())
 {
   case 0:
     ...fai qualcosa...
     break;

   case 1:
     ...fai qualcosa d'altro...
     break;

   case 0:
     ...fai qualcosa di diverso...
     break;

   ....
 }
```

Primitive di comunicazione

La lettura e scrittura di locazioni di memoria remote avviene attraverso due funzioni: `shmem_get()`, `shmem_put()`. Le loro definizioni sono:

```
int shmem_get (long *target, long *source, int len, int pe)
```

```
int shmem_put (long *target, long *source, int len, int pe)
```

Dove:

target :indirizzo dell'array in cui verranno trasferiti i dati;
source :indirizzo dell'array da cui copiare i dati;
len :lunghezza, in long, dell'array;
pe :numero del PE remoto;

Quando una di queste funzioni viene chiamata, il chiamante deve fornire un indirizzo locale al PE chiamante e un secondo indirizzo locale ad un altro PE (chiamato *indirizzo remoto*). Il programmatore deve fare in modo che l'indirizzo remoto sia valido. Nell'uso di dati dichiarati globali o statici il problema non si pone perchè queste variabili vengono allocate allo stesso indirizzo su tutti i PE: questo perchè è il compilatore che, staticamente, decide gli indirizzi di queste variabili. La stessa cosa non vale per variabili di stack o di heap, che su PE diversi possono avere indirizzi differenti. Ciononostante è possibile accedere a dati remoti allocati dinamicamente: la condizione è che vengano allocati con la funzione `shmem_malloc()`. Tale funzione assicura lo stesso indirizzo di una variabile dinamica su tutti i PE.

Primitive di sincronizzazione

Le funzione `shmem_get()` e `shmem_put()` non operano alcun tipo di sincronizzazione. `shmem_get()` ritorna quando il dato remoto è stato caricato nell'array locale (ma il dato remoto potrebbe non essere pronto per essere trasferito). `shmem_put()` ritorna quando i dati sono stati spediti dal processore (ma potrebbero non essere ancora arrivati a destinazione). Il programmatore deve quindi gestire la sincronizzazione tra i processori per evitare problemi di coerenza dei dati.

Questo problema può essere risolto mediante due tecniche: *barriera* ed *evento*. Una barriera è un punto all'interno del programma in cui un PE aspetta che tutti gli altri PE abbiano raggiunto la barriera. Un evento è un punto all'interno del programma in cui tutti i PE sono informati quando un PE ha soddisfatto una certa condizione. Per i nostri scopi sará sufficiente conoscere il meccanismo della barriera.

Il meccanismo della barriera consiste di due parti:

- impostare la barriera;
- aspettare alla barriera finchè tutti i PE non sono arrivati;

Queste azioni vengono rispettivamente svolte dalle funzioni `set_barrier()` e `wait_barrier`.

```
void set_barrier (void)
```

```
void wait_barrier (void)
```

`set_barrier()` indica che il PE chiamante è arriavato alla barriera. `wait_barrier()` blocca il PE chiamante fintantochè tutti gli altri PE non sono arrivati alla barriera.

Per evitare che un PE sprechi troppo tempo aspettando ad una barriera è possibile usare la funzione `test_barrier()`: tale funziona controlla lo stato della barriera e ritorna falso se un qualche PE non è ancora arrivato, vero se tutti i PE hanno impostato la barriera.

```
long test_barrier (void)
```

Una chiamata alla funzione `barrier()` blocca il PE chiamante fintantochè tutti i PE non sono arrivati alla barriera, combinando quindi le funzionalità di `set_barrier()` e `wait_barrier`. La sintassi è:

```
void barrier (void)
```

Questa funzione, oltre a garantire la sincronizzazione garantisce anche il completamento di tutte le scritture remote iniziate da `shmem_put()`.

Coerenza della cache

Questo accade solo su T3D in quanto su T3E la cache viene aggiornata automaticamente dall'hardware.

Quando per spedire dati si usa la funzione `shmem_put()` la cache del PE remoto non viene aggiornata. Quindi c'è la possibilità che il PE remoto possa leggere dalla cache un dato non coerente con la memoria. Bisogna quindi forzare il PE a leggere dalla memoria piuttosto che dalla cache. In gergo viene definita *invalidazione* della cache. La funzione *shmem_udcflush()* invalida l'intera cache del processore:

```
void shmem_udcflush (void)
```

Capitolo 5

Il software

5.1 Introduzione

In questo capitolo si analizzerà il sofware di simulazione approntato con particolare riferimento alle tecniche di implementazione, di gestione dell'I/O, di visualizzazione e di programmazione parallela.

5.2 Progettazione

Nella fase di progettazione del software sono state osservate le seguenti specifiche generali:

- massima portabilità del codice;
- massima efficienza (in tempo di esecuzione);
- modularità;
- massima interoperabilità tra architetture diverse;

Per garantire la massima portabilità si è scelto di codificare in linguaggio ANSI C, oramai uno standard a livello mondiale. Come risultato si è ottenuto che lo stesso codice è stato compilato e usato con successo su architetture diverse: Hewlett-Packard, Cray, Intel. Il tutto senza cambiare una solo riga di codice.

Ottenere la massima efficienza è chiaramente molto difficile ma un attento studio delle operazioni da implementare ha permesso di scartare le soluzioni più inefficienti. Ad esempio tutti i calcoli vengono eseguiti su numeri

interi, non c'è alcuna operazione in virgola mobile. Le prestazioni del software, se comparate con le implementazioni del modello FHP bidimensionale, sono di buon livello ma alcune migliorie sono certamente possibili.

Il software non è stato pensato come una applicazione monolitica ma con una insieme di moduli interagenti. La struttura dei moduli è stata ottimizzata nell'ottica di un uso batch degli stessi, sebbene un uso interattivo sia naturalmente possibile.

La massima interoperabilità è strettamente collegata alla modularità del software: i moduli possono essere eseguiti su architetture diverse, in quanto per comunicare utilizzano un formato di file indipendente dall'architettura. Tale formato è quello fornito dalla libreria HDF sviluppata dal NCSA (National Center for Supercomputing Application).

5.3 Architettura generale

I moduli si possono dividere in due classi:

- moduli pre-elaborazione;
- moduli post-elaborazione;

Tra queste due classi si inserisce il modulo di simulazione, ovvero il programma che simula l'automa cellulare vero e proprio. Il processo per ottenere una simulazione si divide perciò in tre passi:

1. preparazione dati pre-elaborazioni;
2. simulazione;
3. post-elaborazione;

Vediamo i tre passi in dettaglio. A capo di tutto c'è la preparazione di un file di testo, detto *file di risorse*, contenente le risorse o parametri che saranno poi utilizzati nelle fasi successive. Tale fase richiede l'utilizzo di un qualsiasi editor di file di testo in grado di scrivere un file in formato ASCII.

La seconda fase è la generazione del *file di reticolo*, ovvero il file che per ogni nodo del reticolo ne specifica la natura (fluido o ostacolo). Questa fase è gestita dal modulo chiamato `glf`.

La terza fase consiste di un controllo della coerenza dei dati impostati nel file di risorse. Questa fase non è obbligatoria ma è fortemente consigliata in quanto il modulo di simulazione effettua pochi controlli e un parametro

incorretto potrebbe far terminare il programma o portare un allungamento dei tempi di simulazione. Questo modulo si rivela molto utile durante l'utilizzo di sistemi batch, in cui i programmi possono essere eseguiti anche parecchie ore dopo il loro ingresso in coda. Il modulo di controllo è `crcf`.

Quarta fase è la simulazione vera e propria. Il modulo che la implementa è chiamato `ca` ed il suo unico input è il file di risorse. Come output produce un file contenente i dati rilevati nel corso della vita dell'automa con le modalità specificate nel file di risorse.

A questo punto dobbiamo distinguere se il modulo di simulazione è stato eseguito su macchina mono o multiprocessore. Nel primo caso si passa alla prossima fase, nel secondo c'è bisogno di una elaborazione intermedia. Nel caso di esecuzione su sistema multiprocessore è stato preferito che ogni processore producesse il proprio file piuttosto di sincronizzare le scritture su un singolo file. Con questa soluzione si toglie un livello di sincronizzazione molto dispendioso, in quanto l'I/O su disco è notoriamente lento. Quindi il compito qui richiesto è di produrre un unico file di dati fondendo i diversi file scritti dai processori; compito eseguito dal modulo chiamato `merge`.

Il passo successivo consiste nell'operare una media spazio-temporale sui dati scritti dal modulo di simulazione. Quello che si attua è un processo di riduzione del reticolo originario mediando su piccole regioni spaziali (dette macrocelle), al fine di togliere il rumore statistico che di norma affligge gli automi cellulari. Questo compito è svolto dal modulo chiamato `fof`.

Il prossimo passo consiste nell'analisi dei dati della simulazione ed è il punto di arrivo dell'intero processo. Il modulo `av` permette una dettagliata analisi spazio-temporale di tutti i dati della simulazione.

Dopo questa fase si diramano più vie a seconda del tipo di sistema scelto per visualizzare i dati. Il formato di output usato da `av` è piuttosto generico e permette di importare quasi direttamente i dati nei vari applicativi di visualizzazione. La figura 5.1 schematizza l'intero processo appena descritto.

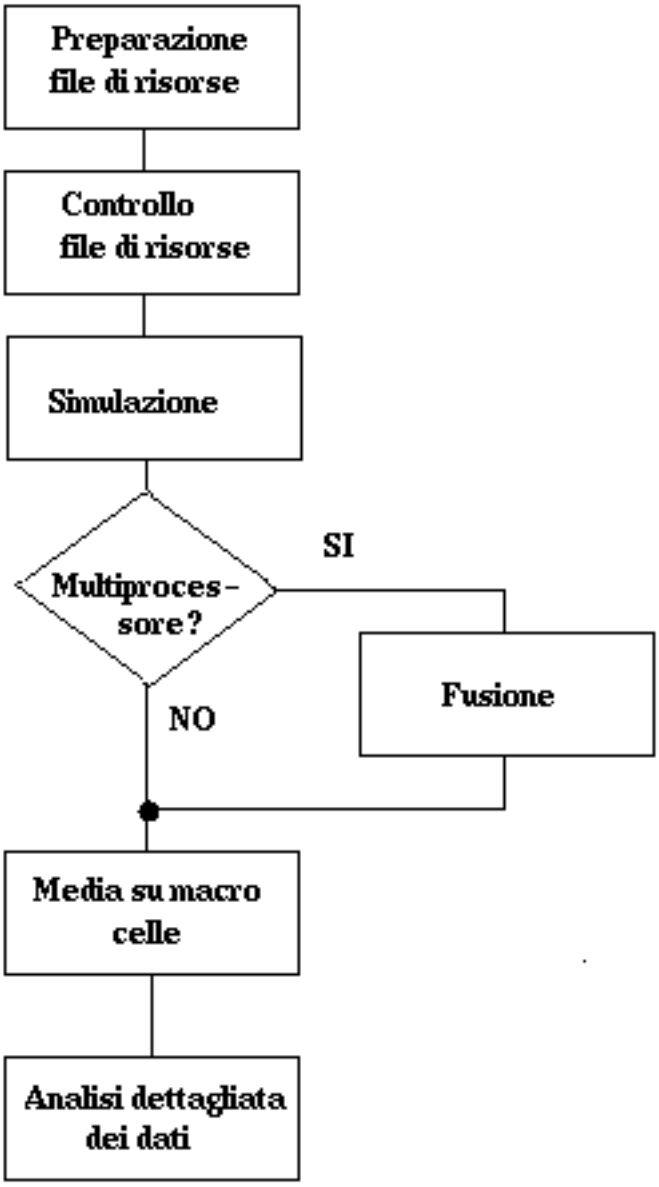

Figura 5.1: il processo di simulazione

5.4 Tecniche di implementazione

Di seguito verranno illustrate le tecniche adottate per ottenere le massime prestazioni dal software di simulazione.

5.4.1 Passo di propagazione

Per il passo di propagazione ci si è affidati ad una tecnica nota in letteratura come *multi-spin coding*. Questa tecnica consiste nel disporre la stessa direzione di più nodi adiacenti nella stessa parola di macchina, coincidente di solito con il tipo `int` del C. Quindi su macchine con interi a 2^k bit ogni intero conterrà la stessa direzione di 2^k nodi adiacenti e occorreranno 12 interi per rappresentare le 12 direzioni di 2^k nodi. I vantaggi di tale tecnica sono:

- nessun spreco di memoria;
- parallelismo intrinseco;

Non vi è alcun spreco di memoria in quanto viene usato esattamente un bit per specificare la presenza o meno di una particella in un data direzione. Si ha un certo grado di parallelismo della propagazione in quanto vengono elaborate più direzioni alla volta, in dipendenza della lunghezza della parola di macchina. Ad esempio su macchine con interi a 64 bit come i Cray, la propagazione avverrà in 64 direzioni contemporaneamente.

Il reticolo dei nodi viene pensato come una serie di piani adiacenti; in realtà poi ogni piano è composto da altri 12 piani, uno per ogni direzione. La struttura dati che implementa il reticolo dei nodi è perciò un array a 3 livelli di indirezione in cui il primo livello identifica il piano su cui si trova il nodo, il secondo identifica la direzione e il terzo identifica l'intero contenente la particella. La figura 5.2 illustra la costruzione della struttura.

Per simulare la simultaneità della propagazione bisogna ricorre certamente ad una struttura di tipo buffer in cui depositare i nuovi stati calcolati. Questa buffer, alla fine dell'aggiornamento, dovrà essere ricopiato nella struttura che implementa il reticolo. Si scarta subito l'ipotesi di un buffer grande quanto la struttura reticolo, in quanto porterebbe ad un notevole consumo di memoria. Una soluzione più efficace risulta osservando che la propagazione in un piano n coinvolge solo i piani $n-1$ e $n+1$: quindi è necessario un buffer grande abbastanza per contenere solo due piani di nodi. Per effetto della struttura di memorizzazione si ha che lo swap tra buffer e reticolo, dopo l'aggiornamento di un piano, si riduce solo allo spostamento di 24 puntatori, indipendentemente dalla grandezza dei piani del reticolo.

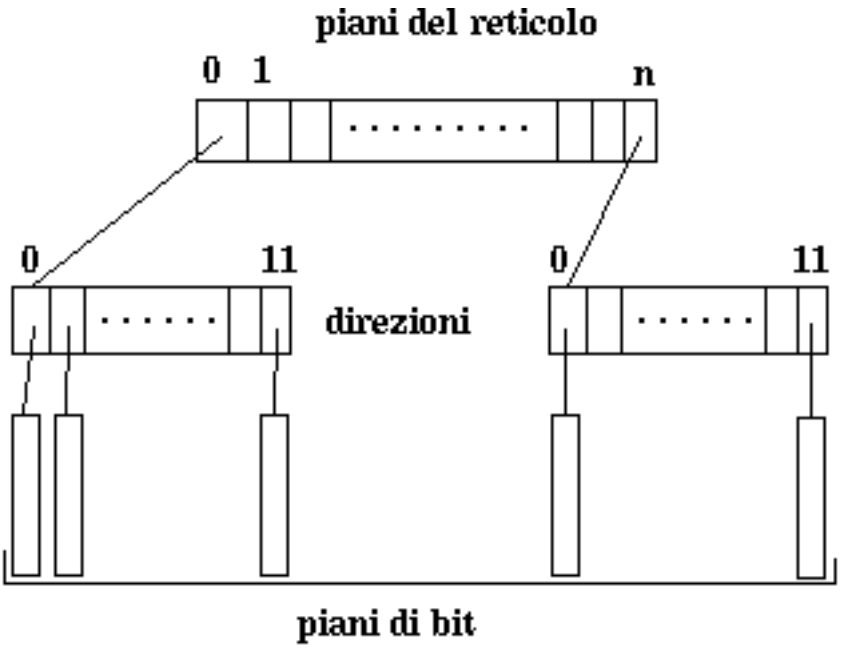

Figura 5.2: memorizzazione del reticolo

5.4.2 Passo di collisione

Il passo di collisione ha richiesto un consistente studio volto alla ricerca di un metodo implementativo efficiente.

In primo luogo si è cercato di applicare al modello rombododecaedrico la codifica multi-spin delle collisioni, come naturale conseguenza dell'applicazione al passo di propagazione. Nella tecnica multi-spin le collisioni sono codificate come funzioni booleane che restituiscono la direzione post-collisione; le funzioni saranno tante quante le possibili direzioni del modello.

Il vantaggio di questa tecnica è, come per la propagazione, il parallelismo intrinseco nell'elaborazione dei nodi. Purtroppo una volta applicata all'insieme di collisioni del modello rombododecaedro, si ha la generazione di funzioni con un numero molto grande di operatori booleani, dell'ordine delle decine di migliaia, anche operando una riduzione degli operatori con appositi programmi di minimizzazione booleana. Se si aggiunge poi il fatto che le funzioni sono 12, si superano i centomila operatori booleani.

Questo studio ha permesso però di trovare una variante della classica tecnica multi-spin per le collisioni.

Innanzitutto si partiziona l'insieme delle collisioni in funzione del numero

di stati di post-collisione possibili per un determinato stato pre-collisione. Per l'automa rombododecaedrico abbiamo le seguenti dimensioni possibili delle classi di collisione: 1, 2, 3, 5, 6, 7, 9, 11, 15, 18, 19, 20, 23, 31.

Costruiamo per ogni dimensione trovata le rispettive funzioni booleane; siano c_i^j queste funzioni, dove i è la dimensione della classe codificata e j è la direzione. Sia a_j^t lo stato della direzione j al tempo t. Allora:

$$a_j^{t+1} = \left[\overline{a_j^t} \wedge (c_1^j \vee c_2^j \vee \cdots \vee c_{31}^j)\right] \vee \left[a_j^t \wedge (c_1^j \wedge c_2^j \wedge \cdots \wedge c_{31}^j)\right]$$

Questa tecnica ha permesso di dimezzare gli operatori booleani necessari, ma il loro numero è in assoluto ancora troppo elevato. Il metodo potrebbe però essere utilizzato nel modello bidimensionale FHP per migliorarne ulteriormente l'efficienza.

La tecnica multi-spin non è stata del tutto abbandonata: le collisioni delle particelle ostacolo sono state implementate secondo questa tecnica. Le collisioni di nodi fluido sono invece state implementate con la classica tecnica della tabella di look-up, in cui per ogni stato dell'automa è associato l'insieme dei possibili stati post-collisione.

La tecnica della tabella di look-up ha il grosso svantaggio di essere estramente sensibile alla densità del fluido impostata. Ovvero le prestazioni risentono molto dei valori di densità impostati.

Infatti la densità condiziona il numero di collisioni possibili per ogni nodo, quindi aumentando la densità aumentano i nodi che possono subire una collisione, con conseguente aumento di tempo macchina. Da alcune prove si è visto che le prestazioni possono anche peggiorare di un fattore 2, rispetto alla massima velocità rilevata.

5.5 Raccolta dei dati

Nell'arco della vita dell'automa cellulare è possibile raccogliere informazioni sullo stato dell'automa. Le informazioni che si possono rilevare sono:

- densità, intesa come numero di particelle per nodo;
- quantità di moto.

Queste quantità sono misurate *nodo per nodo*, durante la simulazione non viene eseguito alcun processo di media.

Nell'ottica delle massime prestazioni, queste quantità sono gestite tramite numeri interi e tutto il processo di raccolta dati coinvolge solo operazioni su interi. La chiave di tutto stà nella trasformazione delle coordinate dei

velocity vectors in coordinate intere. Tale trasformazione è possibile grazie alla matrice M:

$$M = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Per la densità non c'è alcun problema dato che è già di per sè un numero intero.

Per utilizzare poi i dati della quantità di moto bisognerà quindi trasformarli in coordinate reali. Questo si attua facilmente con la matrice M' inversa di M:

$$M' = \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & \sqrt{2} & 0 \\ 0 & 0 & 1/2 \end{pmatrix}$$

Input/Output

La decisione di raccogliere i dati nodo per nodo ha dalla sua la possibilità di una più fine post-elaborazione dei dati, ma ha lo svantaggio di richiedere molto spazio su disco.

In particolare le quantità vengono codificate con interi con segno a 16 bit quindi per ogni nodo del reticolo sono richiesti 8 byte per una misurazione.

Il formato adottato per i dati su disco è quello fornito dalla libreria HDF (Hierarchical Data Format) sviluppata dal NCSA (National Center for Supercomputing Applications), University of Illinois. La libreria è disponibile in rete sotto forma di sorgente e si stà rapidamente imponendo come standard per la memorizzazione dei dati scientifici.

Tra i servizi offerti evidenziamo in particolare la completa indipendenza del formato dalle varie piattaforme hardware: questo permette ad esempio di eseguire le simulazioni su macchine Cray a 64 bit e poi gestire la fase di post-elaborazione su normali macchine Intel a 32 bit.

Un'altra caratteristica molto utile è la possibilità di comprimere i dati prima di scriverli. La compressione è implementata con gli algoritmi più moderni ed è totalmente trasparente all'utente. Da alcuni test è risultato che i file compressi sono mediamente 4-5 volte più corti di quelli non compressi. Questo ha quindi permesso di risolvere in parte il problema del grande spazio necessario su disco.

Il formato HDF viene usato tra il modulo di simulazione `ca` e il modulo di media su macrocella `fof` e tra `fof` e il modulo di analisi `av`.

5.6 Programmazione parallela

L'algoritmo di simulazione è stato sviluppato fin dall'inizio tenendo conto del fatto che doveva essere adattabile anche a macchine parallele. Quindi la struttura dati introdotta nella sezione (5.4) è stata pensata proprio sulla base di questa necessità.

La parte parallela del software, pur essendo non molto estesa, ha richiesto una fine messa a punto, allo scopo di ottenere le massime prestazioni dai sistemi Cray MPP.

5.6.1 Divisione del lavoro

L'elaborazione dell'intero reticolo viene spezzata su più processori assegnando ad ogni processore una porzione di reticolo; la divisione del reticolo avviene per piani ed è gestita automaticamente dal programma: l'utente deve solo specificare quanti processori usare per eseguire l'applicazione. Logicamente i processori sono collegati ad anello, come si può vedere dalla figura 5.3.

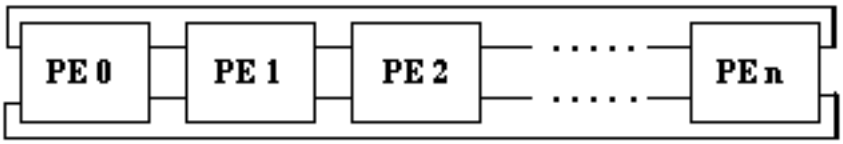

Figura 5.3: schema di comunicazione

Dopodichè ogni processore elabora in locale il proprio reticolo e dopo aver eseguito la propagazione e la collisione c'è una sincronizzazione tra tutti i processori per lo scambio dei piani intercomunicanti. Quindi c'è una comunicazione ad ogni iterazione della simulazione.

Lo scambio dei piani avviene attraverso due buffer di comunicazione allocati con `shmem_malloc()`. In questi buffer vengono copiati i piani da spedire, dopodichè con una `shmem_put()` vengono spediti in blocco al processore destinatario. Dopo i dati viene spedito anche un segnale di sincronizzazione, per informare il destinatario dell'avvenuto arrivo. I piani vengono spediti in blocco perchè risulta molto più conveniente fare pochi grossi trasferimenti.

Per la comunicazione vengono usate le put perchè sul T3D, il primo sistema Cray MPP, erano più veloci delle get, quindi si è deciso di usare le put anche se richiedono un utilizzo più accorto. Sul nuovo T3E get e put hanno la stessa velocità ma anche in questo caso è preferibile usare le put,

in quanto sono routine non bloccanti, cioè il processore non viene bloccato in attesa che i dati arrivino a destinazione.

Un problema con l'utilizzo della put su T3E è causato dall'adozione di un algoritmo di routing adattivo dinamico: il sistema non garantisce che l'ordine di emissione di due o più put corrisponda all'ordine di arrivo dei dati. A causa di ciò bisogna inframezzare due chiamate consecutive a `shmem_put()` con una chiamata a `shemem_fence()`, per forzare il completamento come da ordine di emissione. Sul T3D questo problema non sussiste, in quanto l'algoritmo di routing è di tipo statico.

5.6.2 Sincronizzazione

Una prima sincronizzazione avviene all'inizio di ogni iterazione, con una chiamata a `barrier()`. Una volta raggiunta la barriera tutti i i processori si spediscono i piani intercomunicanti e il segnale di sincronizzazione con delle `shmem_put()`. Dopo ciò ogni processori attende il segnale di sincronizzazione dai suoi due vicini.

Il flag di sincronizzazione è implementato tramite due variabili intere definite globali, così da essere remotamente accessibili. I flag sono due perchè ogni processore comunica con al massimo due altri processori.

Un processore comunica al suo vicino di aver spedito i piani scrivendo un valore diverso da zero in queste variabili. Quindi ogni processore deve attuare un controllo ciclico su queste variabili.

5.6.3 Coerenza della cache

Sul T3E la cache è mantenuta coerente direttamente dall'hardware. Sul T3D è invece il programmatore che deve preoccuparsi di ciò, invalidando la cache con una chiamata a `shmem_udcflush()`.

Nel nostro caso si deve invalidare la cache solo prima della lettura dai buffer di comunicazione dei dati appena arrivati.

5.6.4 Input/Output

Ogni processore genera un suo file di output in cui scrive i dati in maniera indipendente dagli altri processori, per evitare il formarsi di un grosso collo di bottiglia. Tutti i file di output verranno poi fusi in unico file in modo tale da mascherare la struttura parallela della macchina.

Questa gestione parallela dell'I/O rispecchia in qualche modo la struttura dell'I/O del Cray T3E: infatti ogni quattro PE c'è una interfaccia alla rete di I/O, consentendo quindi più operazioni di I/O simultanee.

Oltre al file di output ogni processore apre un file di log in cui scrivere, in caso di errore, dei messaggi diagnostici. Se la simulazione arriva al termine normalmente questi file verrano cancellati automaticamente dai PE che li avevano aperti. Alla fine resterà solo il file di log del PE 0, incaricato di misurare le prestazione del software.

Capitolo 6

Risultati e sviluppi futuri

6.1 Introduzione

In questo capitolo si illustreranno i risultati di alcune simulazioni eseguite con il software approntato. In particolare si è cercata una conferma della distribuzione all'equilibrio ed una conferma del moto laminare.

6.2 Distribuzione all'equilibrio

In questa parte si è cercato di verificare la validità della espansione di Chapman-Enskog (3.5) per l'automa cellulare rombododecaedrico. In realtà non si considera tutta l'espansione ma ci si ferma al termine lineare, cioè si considera:

$$f_a = f(1 + c^{(1)}\vec{e}_a \cdot \vec{u}) \tag{6.1}$$

Essendo la precedente l'espressione di una probabilità deve succedere che $f_a \in [0, 1]$. Questo porta ad identificare dei limiti nei valori della variabili macroscopiche $\vec{u}$ ed n. Infatti:

$$f_a = f(1 + c^{(1)}\vec{e}_a \cdot \vec{u}) = f(1 + c^{(1)}|\vec{e}_a||\vec{u}|\cos\theta)$$

ma $|\vec{e}_a| = 1\ \forall a$ quindi:

$$f_a = f(1 + c^{(1)}|\vec{u}|\cos\theta) \tag{6.2}$$

Studiamo quando $f_a > 0$. Per la (6.2) deve essere:

$$f(1 + c^{(1)}|\vec{u}|\cos\theta) > 0$$

quindi:

$$|\vec{u}| \cos\theta) > -\frac{1}{c^{(1)}}$$

Il caso peggiore si ha quando $\cos\theta = -1$:

$$|\vec{u}| < \frac{1}{c^{(1)}}$$

Nel nostro caso $c^{(1)}$ vale 3, quindi abbiamo ottenuto la limitazione:

$$|\vec{u}| < \frac{1}{3} \tag{6.3}$$

Analizziamo ora quando $f_a < 1$. Sempre per la (6.2) deve essere:

$$f(1 + c^{(1)}|\vec{u}| \cos\theta) < 1$$

ovvero:

$$|\vec{u}| \cos\theta < \frac{1-f}{fc^{(1)}}$$

il termine di destra della precedente disuguaglianza è sempre positivo: quindi il caso peggiore si ha quando $\cos\theta = 1$. Si ha perciò:

$$|\vec{u}| < \frac{1-f}{fc^{(1)}}$$

Per il modello rombodecaedrico $f = \frac{n}{12}$, dove n è la densità particellare. La precedente diventa quindi:

$$|\vec{u}| < \frac{4}{n} - \frac{1}{3} \tag{6.4}$$

Per questo prova si è eseguita una simulazione con reticolo di base quadrata di 32 nodi per lato e lunghezza di 64 nodi. Il file di reticolo è stato impostato con tutti nodi fluido: cioè si sono imposte le condizioni al contorno periodiche. La simulazione ha avuto una lunghezza di 15000 iterazioni, scartando le prime 10000 considerate come transitorio. I parametri fluidodinamici impostati erano: velocità 0.04 e densità 10.7. Il modulo della velocità mediato su tutta la simulazione e su tutto il reticolo è stata di 0.4271, quindi con errore di circa il 5% rispetto al valore impostato. La densità misurata è invece risulta essere 10.6897, quindi molto vicina al valore impostato.

L'errore relativamente alto della velocità non dovrebbe spaventare più di tanto poichè è equamente distribuito su tutti i nodi del reticolo: infatti non ci sono zone ad alta velocità e zone a bassa velocità.

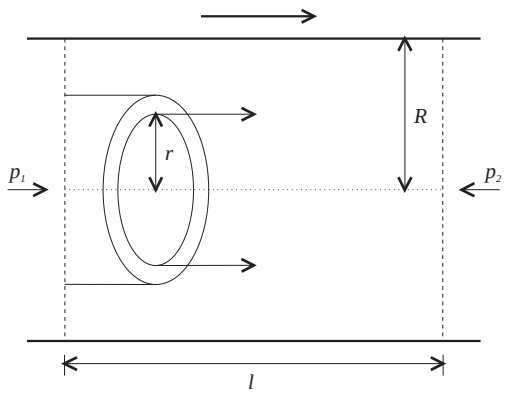

Figura 6.1: moto di Poiseuille

6.3 Flusso di Poiseuille

Come è noto il moto di un fluido viscoso all'interno di un condotto cilindrico è retto dalla legge di Poiseuille. Se il condotto ha certo raggio R allora si può dimostrare che il moto del fluido può essere descritto dal moto di lamine cilindriche che scorrono una dentro l'altra coassiali al condotto [SB92]. Se il moto è stazionario la velocità della generica lamina di raggio r è:

$$v(r) = \frac{\Delta p}{4\nu l}[R^2 - r^2]$$

Dove $\Delta p = p_1 - p_2$ è la differenza di pressione tra la parte a monte e a valle del condotto, l è la lunghezza del condotto e ν è la viscosità del fluido. Dalla legge risulta quindi che il profilo di velocità all'interno del condotto è di tipo parabolico. La situazione è illustrata nella figura 6.1.

6.3.1 Condotto non cilindrico

Le prime prove sono state effettuate su condotti non cilindrici ma a forma di parallelepipedo. Questo perchè il reticolo per essere elaborato deve per forza essere di questa forma.

Una simulazione é stata eseguita con un reticolo di base quadrata di 64 nodi e lunghezza di 128; il file di reticolo è stato generato in modo tale da porre nodi ostacolo sulle pareti del condotto. La simulazione è durata 20000 iterazioni, con un transitorio impostato di 10000 iterazioni. In figura 6.2 è visualizzato il modulo della velocià mediato su tutta la simulazione e lungo l'asse di scorrimento del fluido: come si vede il profilo generato è verosimilmente parabolico. Nella stessa figura e nella figura 6.3 vengono visualizzate

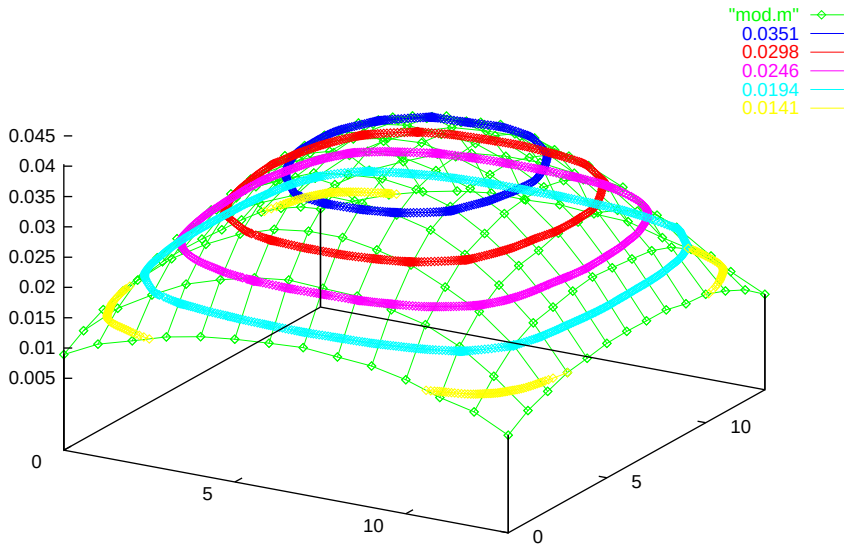

Figura 6.2: velocità su tutta la simulazione

le curve di livello. Come si può vedere il fluido ha effettivamente un moto laminare. Le lamine chiaramente sono potranno essere circonferenze in quanto il condotto non è cilindrico.

La velocità impostata era 0.04 con una densità di 10.7 particelle per nodo. L'iniezione di particelle è a profilo piatto in ingresso e a profilo parabolico in uscita. La velocità media in tutto il condotto è risultata essere di 0.2564 mentre la velocità del filetto fluido centrale, la parte più interessante, è risultata essere di 0.39985. Quindi in buon accordo con il valore impostato. La densità media misurata in tutto il condotto è stata di 10.9168, quindi con errore del 2 % circa rispetto al valore impostato: un errore tutto sommato accettabile. In figura 6.4 è visualizzato il profilo della densità mediato su tutta la simulazione e lungo l'asse di scorrimento del fluido. Si nota, come per il modulo della velocità un profilo parabolico, sebbene molto meno marcato.

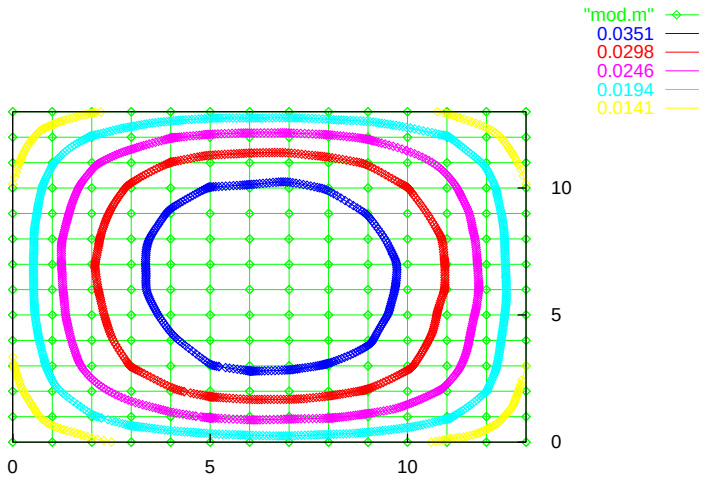

Figura 6.3: velocità: curve di livello

6.3.2 Condotto cilindrico

Per questa prova si è simulato un condotto cilindrico inserendo nel reticolo degli ostacoli lungo tutte le pareti del condotto. La figura 6.5 mostra una sezione del reticolo elaborato.

Per questa simulazione è stato scelto un reticolo a base quadrata di lato 96 nodi e lunghezza 256 nodi, quindi un reticolo di oltre due milioni di nodi. L'iniezione è a profilo piatto in ingresso e a profilo parabolico in uscita. Il numero totale di iterazioni è 20000, con un transitorio di 10000. I parametri fluidodinamici velocità e densità sono gli stessi del precedente caso.

La velocità misurata mediata su tutta la simulazione e su tutto il reticolo è risultata 0.34486 mentre la velocità del filetto fluido centrale è risultata 0.42406. Questo equivale ad un errore di circa il 5% rispetto alla velocità impostata ed è dovuto principalmente alla strozzatura del condotto in ingresso.

La densità misurata su tutta la simulazione è stata di 10.8686 particelle per nodo: l'errore è perciò dell'1.5 % circa.

In figura 6.6 è mostrato il modulo della velocità mediata lungo l'asse

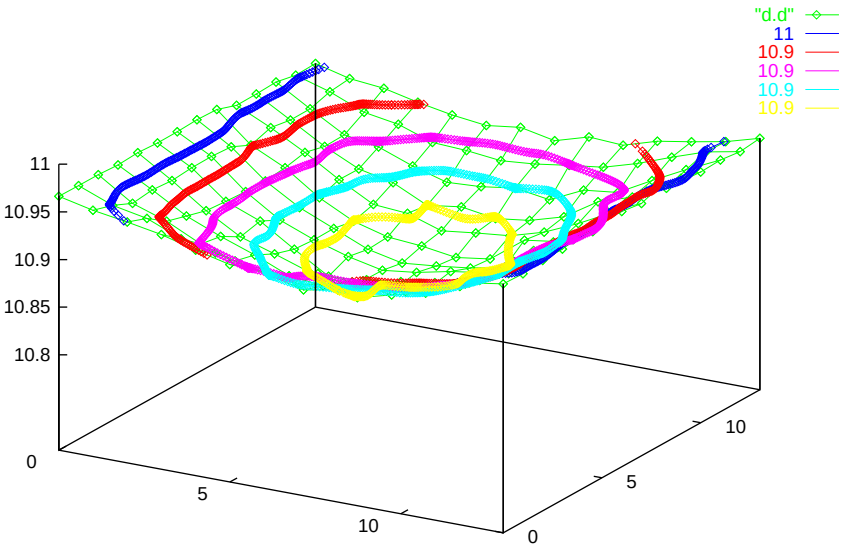

Figura 6.4: densità su tutta la simulazione

di scorrimento del fluido. Si nota che il profilo, come ci si aspettava, è di tipo parabolico. Le curve di livello indicano chiaramente che il moto è di tipo laminare. La figura 6.7 visualizza le curve di livello per il modulo della velocità e permette di osservare il fenomeno da una migliore angolazione. Si nota come le curve si stiano approssimando sempre più a delle circonferenze.

Per la densità è stato osservato lo stesso fenomeno del condotto non cilindrico: un profilo parabolico convesso, cioè con una depressione a centro reticolo. La figura 6.8 visualizza la densità mediata lungo l'asse di scorrimento.

6.4 Prestazioni

Il software di simulazione permette di aggiornare circa 400000 nodi/s su CPU Intel Pentium a 100MHz. La prestazione si intende comprensiva di tutte le fasi di simulazione: propagazione, collisione e rilevamento dati. Per nodi si intendono celle di 12 particelle ciascuna: quindi la velocità di aggiornamento

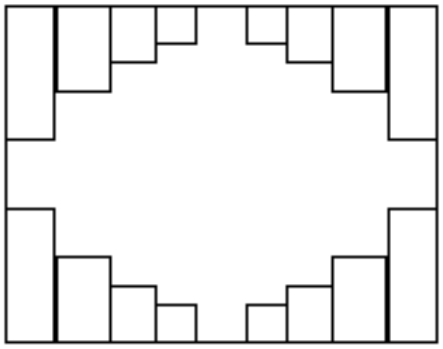

Figura 6.5: approssimazione del condotto cilindrico

è di 4.800.000 celle booleane al secondo. Tutto sommato si tratta di un buon risultato considerando la classe del processore usato.

Da prove effettuate sul sistema Cray T3E sono emerse delle prestazioni ancor piú sorprendenti. Con 128 processori si è raggiunta la velocità di 140 milioni di nodi/s, un risultato paragonabile a quello ottenibile con architetture dedicate. Inoltre le prestazioni del software sono completamente scalabili rispetto al numero di processori usati: cioè un aumento di un fattore k del numero dei processori disponibili corrisponde ad un aumento di un fattore k delle prestazioni.

6.5 Sviluppi futuri

Lo svolgimento rigoroso dei test fluidodinamici classici quale ad esempio il Backward Facing Step rientra sicuramente negli sviluppi futuri. Gli strumenti software sono già approntati e si tratta quindi di effetturare misure precise e confrontarle con gli esperimenti pratici.

Il software di simulazione può essere migliorato per quanto riguarda la tolleranza agli errori di sistema. Si può pensare infatti di operare a intervalli di tempo regolari un salvataggio su disco dello stato dell'automa, in modo tale che una caduta del sistema non obblighi a ricominciare dall'inizio la simulazione. Una possibile realizzazione di questa idea potrebbe essere quella di introdurre un *file di stato* da cui il software di simulazione legga lo stato iniziale del reticolo.

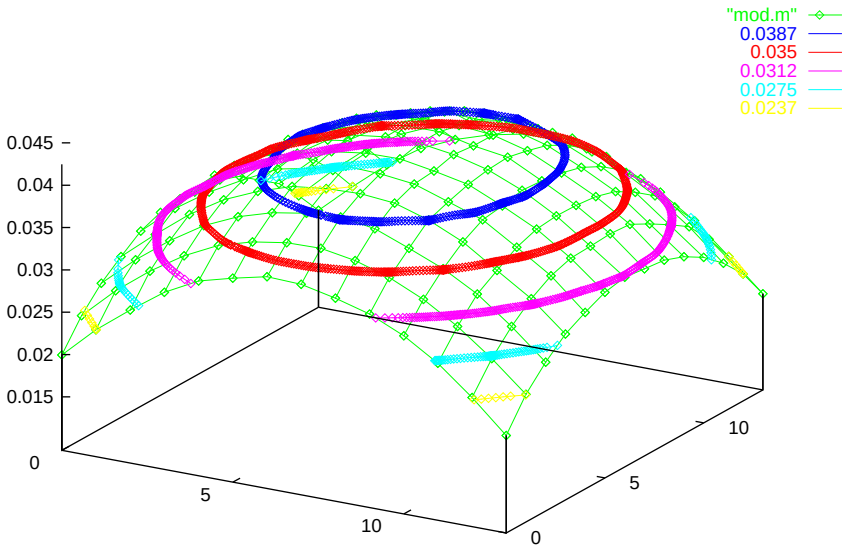

Figura 6.6: modulo della velocità (condotto cilindrico)

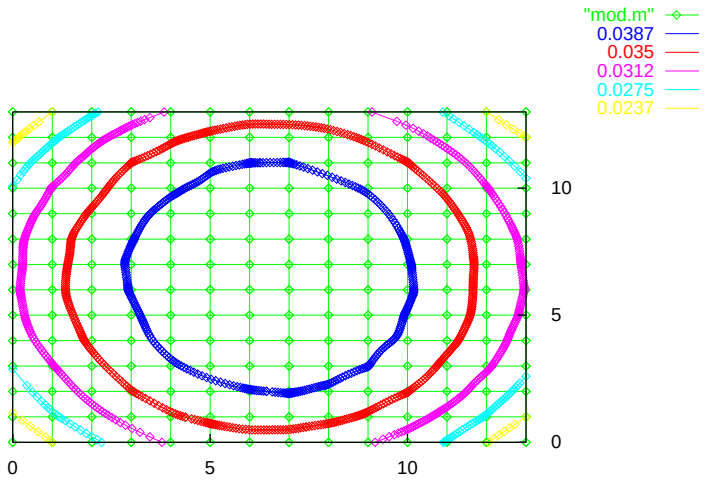

Figura 6.7: curve di livello (condotto cilindrico)

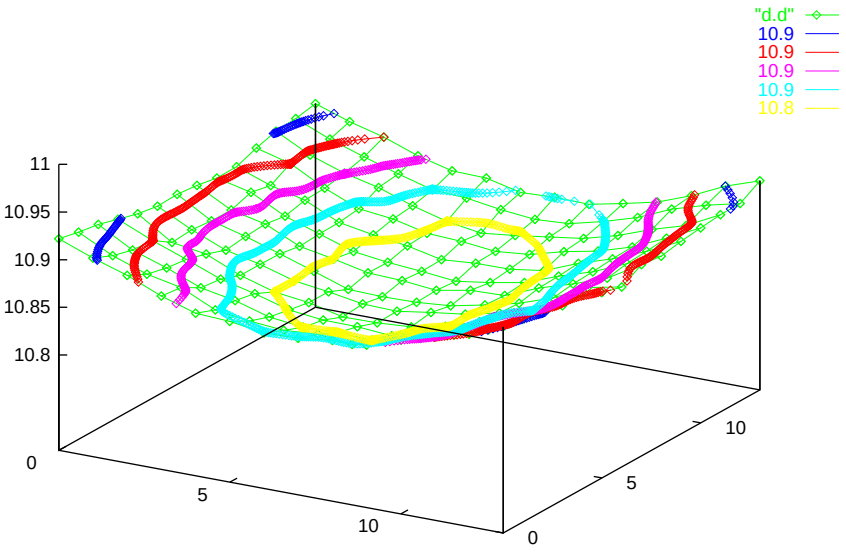

Figura 6.8: densità (condotto cilindrico)

Appendice A

Rotazioni degli assi e matrici

Consideriamo le trasformazioni di coordinate causate da una arbitraria rotazione rigida del sistema di riferimento.

Consideriamo un sistema di riferimento $O123$ nel quale le coordinate di un punto P sono x_1, x_2 e x_3 e un nuovo sistema di riferimento $O\overline{123}$ in cui le coordinate dello stesso punto siano $\bar{x}_1$, $\bar{x}_2$ e $\bar{x}_3$. Sia l_{ij} il coseno dell'angolo compreso tra Oi e $O\bar{j}$, $i, j = 1, 2, 3$. Allora l_{1j}, l_{2j}, l_{3j} sono i coseni delle direzioni $O\bar{j}$ nel vecchio sistema e l_{i1}, l_{i2}, l_{i3} sono i coseni di Oi nel nuovo.

Per definizione la nuova coordinata $\bar{x}_j$ è la lunghezza della proiezione del segmento $\overline{OP}$ sull'asse $O\bar{j}$, cioè:

$$\bar{x}_j = l_{1j}x_1 + l_{2j}x_2 + l_{3j}x_3 \qquad j = 1, 2, 3 \tag{A.1}$$

Alternativamente lavorando dal nuovo sistema al vecchio, x_i è la proiezione di $\overline{OP}$ su Oi perciò:

$$x_i = l_{i1}\bar{x}_1 + l_{i2}\bar{x}_2 + l_{i3}\bar{x}_3 \qquad i = 1, 2, 3 \tag{A.2}$$

Se componiamo i numeri l_{ij} in una matrice:

$$L = \begin{pmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{pmatrix}$$

e le coordinate nei vettori colonna x e $\bar{x}$ l'equazione (A.2) si può riscrivere come:

$$x = L\bar{x} \tag{A.3}$$

Sia ora L' la trasposta di L, allora l'equazione (A.1) diventa:

$$\bar{x} = L'x \tag{A.4}$$

Bibliografia

[dL88] Dominique d'Humierès and Pierre Lallemand. Numerical simulations of hydrodynamics with lattice gas automata in two dimensions. In Gary Doolen et al., editor, *Lattice Gas Methods for Partial Differential Equations*, pages 297–332. Addison Wesley, 1988.

[DMDH88] A. Despain, C. E. Max, G. Doolen, and B. Hasslacher. Prospects for a lattice-gas computer. In Gary Doolen et al., editor, *Lattice Gas Methods for Partial Differential Equations*, pages 211–218. Addison Wesley, 1988.

[ea90] Gary Doolen et al., editor. *Lattice Gas Methods for Partial Differential Equations*. Addison Wesley, 1990.

[FdH$^+$88] Uriel Frisch, Dominique d'Humieres, Brosl Hasslacher, Pierre Lallemand, Yves Pomeau, and Jean-Pierre Rivet. Lattice gas hydrodynamics in two and three dimensions. In Gary Doolen et al., editor, *Lattice Gas Methods for Partial Differential Equations*, pages 75–135. Addison Wesley, 1988.

[FHP88] Uriel Frisch, Brosl Hasslacher, and Yves Pomeau. Lattice-gas automata for the Navier-Stokes equation. In Gary Doolen et al., editor, *Lattice Gas Methods for Partial Differential Equations*, pages 11–18. Addison Wesley, 1988.

[FHS89] M. Mandal F. Hayot and P. Sadayappan. Implementation and performance of a binary lattice gas algorithm on parallel processor system. *Journal of Computational Physics*, 80:277–287, 1989.

[FTW84] D. Farmer, T. Toffoli, and S. Wolfram, editors. *Cellular Automata*, North Holland, 1984.

[HdPP76] J. Hardy, O. de Pazzis, and Y. Pomeau. Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions. *Physical Review A*, 13:1949–1960, 1976.

[Hen88] Michel Henon. Viscosity of a lattice gas. In Gary Doolen et al., editor, *Lattice Gas Methods for Partial Differential Equations*, pages 181–207. Addison Wesley, 1988.

[Hen92] Michel Henon. Implementation of the fchc lattice gas model on the connection machine. *Journal of Statistical Physics*, 68:353–378, 1992.

[Hun46] Rouse Hunter. *Elementary Mechanics of Fluid*. Dover Publications, Inc., 1946.

[Lig86] James Lighthill. *An informal introduction to theoretical fluid mechanics*, volume 2 of *IMA Monograph Series*. Clarendon Press, Oxford, 1986.

[Rut89] A. Rutherford. *Vectors, tensors and the basic equation of fluid mechanics*. Dover Publications, 1989.

[Sal94] A. Sala. Fluidodinamica tridimensionale con automi cellulari: teoria e simulazione vettoriale. Dipartimento Scienze dell'Informazione, Milano, 1994. Tesi di Laurea.

[SB92] V.R. Manfredi S.A. Bonometto, F.B. Lucchin. *Elementi di fisica generale*. Edizioni Libreria Cortina, 1992.

[Sbr94] D. Sbragion. Simulazioni di fluidodinamica tramite automi cellulari bidimensionale. Dipartimento Scienze dell'Informazione, Milano, 1994. Tesi di Laurea.

[Tri88] D. J. Tritton. *Physical Fluid Dynamics*. Clarendon Press, Oxford, 1988.

[Wol86] Stephen Wolfram. Cellular automaton fluids 1: Basic theory. *Journal of Statistical Physics*, 45:471–526, 1986.