Politecnico di Milano

Dipartimento di Scienze e Tecnologie Aerospaziali
Corso di Laurea Magistrale in Ingegneria Aeronautica

Simulazione numerica diretta della fluidodinamica nasale: aspetti di efficienza computazionale

Relatore:
Prof. Maurizio QUADRIO

Correlatore:
Prof. Franco AUTERI

Autore:
Marco LUGARESI
Matr. 894179

Anno Accademico 2018 - 2019

La creatività è contagiosa, trasmettila.

Albert Einstein

Sommario

Il lavoro presentato nasce all'interno di un progetto più ampio noto come *OpenNose* che vede la collaborazione del Politecnico di Milano e del reparto di otorinolaringoiatria dell'ospedale San Paolo di Milano, con l'obiettivo di sviluppare un metodo ingegneristico per fornire supporto diagnostico ai chirurghi ed incrementare il tasso di successo degli interventi sulle vie aeree superiori. Negli anni è stata messa a punto dal gruppo di ricerca una procedura che, a partire dai dati degli esami di routine effettuati sui pazienti, fornisce la soluzione del problema fluidodinamico e la utilizza come punto di partenza per l'indagine diagnostica, affidata alle tecniche di intelligenza artificiale. A causa della complessità geometrica delle cavità nasali e dell'accuratezza necessaria per ottenere un risultato chirurgicamente accettabile e grazie al ridotto numero di Reynolds del problema, le simulazioni utilizzate sono DNS che, a causa della risoluzione richiesta, risultano il collo di bottiglia della procedura. L'obiettivo di questa tesi è indagare quanto sia possibile accelerare la procedura risolutiva sostituendo l'attuale solutore basato su OpenFOAM. Come strumento di confronto si è utilizzato un codice DNS basato sull'idea del professor Guermond che consente di ottenere il risultato delle equazioni incomprimibili di Navier-Stokes risolvendo unicamente problemi monodimensionali che tradotti in sistemi lineari tridiagonali possono essere risolti in modo molto efficiente. Tale algoritmo, in linea di principio limitato alle griglie cartesiane, viene esteso a geometrie arbitrarie utilizzando il sempre più diffuso metodo dei contorni immersi che, attraverso opportune forzanti, permette di introdurre gli effetti geometrici nelle equazioni del moto. Il risultato è un incremento significativo di efficienza computazionale con uno speed-up seriale che, nei limiti del lavoro presentato, supera il fattore dieci.

Parole chiave: CFD, vie aeree superiori, DNS, metodo dei contorni immersi, splitting dimensionale

Abstract

The work presented belongs to a larger project called *OpenNose* based on the collaboration of the Politecnico di Milano and the otolaryngology department of San Paolo hospital in Milan, with the aim of developing an engineering method to provide diagnostic support to surgeons and increase the success rate of interventions on the upper airways. Over the years, a procedure has been developed by the research team which, starting from the data of routine tests carried out on patients, provides the solution of the fluid-dynamic problem and uses it as a starting point for the diagnostic investigation entrusted to artificial intelligence techniques. Due to the geometric complexity of the nasal cavities and the accuracy necessary to obtain a surgically acceptable result and thanks to the reduced Reynolds number of the problem, DNS simulations has to be used and due to the high cost, rapresent the bottleneck of the procedure. The aim of this thesis is to investigate how much it is possible to speed up the solution procedure by replacing the current Open FOAM based solver. As a comparison tool, a DNS code based on the idea of Professor Guermond was used which allows to obtain the result of Navier-Stokes incompressible equations by solving only tridiagonal linear systems for which very efficient solution techniques are available. This idea, in principle limited to Cartesian grids, is extended to arbitrary geometries using the increasingly popular immersed boundary methods which, through suitable forcing, allows to introduce geometric effects into the equations of motion. The result is a significant increase in computational efficiency with a serial speed-up which, within the limits of the work presented, exceeds the factor ten.

Key-words: CFD, upper airways, DNS, immersed boundary, dimensional splitting

Ringraziamenti

Desidero ringraziare tutti coloro che, in modo più o meno diretto, hanno dato un contributo al compimento di questo lavoro. Per prima cosa un ringraziamento speciale va al Professor Quadrio che, nonostante le difficoltà incontrate nello sviluppo della tesi e i limiti imposti dal periodo storico, ha sempre trovato il modo di aiutarmi dimostrando disponibilità, interesse e passione. Un grazie sentito anche al Professor Auteri che, nonostante i numerosi impegni, è riuscito a fornire un contributo operativo fondamentale per lo svolgimento del lavoro. Ringrazio l'ingegner Schillaci che nei, seppur rari, momenti di necessità si è reso disponibile. Un grazie a tutto il gruppo di ricerca con cui ho avuto il piacere di condividere quasi un anno di lavoro e in particolare a Gabriele che oltre ad aiutarmi quando necessario ha reso più facile l'integrazione e più interessante la permanenza.
Un ringraziamento speciale al Professor Quartapelle che nell'ultimo anno di università mi ha permesso di venire a contatto con aspetti del mondo scientifico spesso trascurati e ignorati. Un grazie particolare per i consigli, l'immensa disponibilità ad ascoltare e le sue parole che sono risultate fonte di ispirazione per la mia crescita personale.

Un profondo ringraziamento ai miei genitori che in modo molto diverso ma parimente fondamentale hanno saputo aiutarmi, sostenermi e indirizzarmi senza mai ostacolare le mie scelte. Un grazie speciale a mia sorella Marina che con poca pazienza ma grande disponibilità ha risposto alle infinite domande di carattere medico senza mai tirarsi indietro. Un grazie particolare anche a mio fratello Mattia che da sempre è sicuramente il mio più grande sostenitore. Grazie anche a Meg per avermi tenuto al lavoro a suon di sgridate. Grazie anche ai miei nonni e a tutti i parenti senza i quali non sarei certamente lo stesso.

Un ringraziamento a Bryan, Filippo, Fulvio, Alice, Emilia e tutti gli amici che hanno condiviso con me gran parte della vita aiutandomi a crescere e affrontare con gioia e serenità ogni giorno degli anni trascorsi. Un ringraziamento speciale a Vanilla, Ludovica e Chiara per avermi sopportato e sostenuto in un momento difficile. Grazie anche a Federica e Alessia per esserci sempre state nei momenti del bisogno.
Un profondo ringraziamento a Veronica, compagna di viaggio e fonte di ispirazione lun-

go la maggior parte di questo percorso. Grazie per il tuo sostegno e la tua fantasia.

Grazie a tutti gli amici che hanno condiviso con me questo bellissimo percorso universitario: grazie a Francesco e Lorenzo con i quali fin dal primo giorno abbiamo creato qualcosa di insostituibile, grazie a Silvia e Francesca che, in periodi diversi, mi hanno aiutato a "vedere" la lavagna, grazie a Simone e tutti gli altri per le indimenticabili esperienze che hanno reso così piacevoli questi anni al Politecnico. Un grazie a Giovanni e Sebastiano due tra i pochi compagni la cui opinione ingegneristica ho sempre apprezzato, grazie a Roberto, Alessio, Andrea e tutti i compagni, divenuti amici, dei vari lavori di gruppo. Grazie a Guido per avermi fatto compagnia esame dopo esame in questo lungo percorso ed essere diventato un grande amico con cui è sempre piacevole confrontarsi.
Grazie infine ad Alessandro, Andrea e a tutto il gruppo per avermi tenuto compagnia e fatto sorridere in un brutto momento; grazie agli amici più recenti e a tutte le persone che in diversi modi hanno contribuito a rendere migliore questo periodo della mia vita.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

Il lavoro presentato nasce in un progetto più ampio noto come *OpenNose* che vede la collaborazione multidisciplinare di più istituzioni con l'obiettivo di fornire assistenza diagnostica e procedurale, basata su tecniche ingegneristiche quali la fluidodinamica computazionale e il machine learning, ai chirurghi che devono operare sulle vie aeree superiori. Le aree di competenza principalmente coinvolte nel progetto sono la fluidodinamica computazionale affidata al dipartimento di scienze e tecnologie aerospaziali del Politecnico di Milano, l'implementazione delle tecniche di intelligenza artificiale per l'interpretazione dei dati ottenuti, gestita principalmente dal dipartimento di elettronica, informazione e bioingegneria del Politecnico di Milano e lo studio dell'anatomia e delle patologie di interesse affidato al reparto di chirurgia otorinolaringoiatra dell'ospedale San Paolo di Milano. Questi tre gruppi collaborano a stretto contatto in modo che ciascun membro del gruppo sia consapevole sia della direzione generale di ricerca indicata principalmente dalle necessità del reparto medico che delle soluzioni ingegneristiche proposte e del loro potenziale.

1.1 Inquadramento generale

In questo contesto negli anni è stata messa a punto una procedura che a partire dai dati ottenuti con esami di routine sui pazienti fornisce la soluzione del problema fluidodinamico nella complessa geometria delle cavità nasali. In seguito questo risultato viene elaborato ed utilizzato per l'indagine diagnostica, ancora in via di sviluppo, affidata alle tecniche di intelligenza artificiale. Più in dettaglio a partire dalla TAC (Tomografia Assiale Computerizzata) del paziente, si ottiene un file stl che viene utilizzato, grazie al software OpenFOAM, per generare una mesh e risolvere il problema fluidodinamico di interesse. L'obiettivo di questa tesi è indagare quanto ad oggi OpenFOAM risulti essere lo strumento adatto per la soluzione di tale problema. Infatti seppur esso sia uno

degli strumenti più diffusi per la soluzione di correnti incomprimibili, gli algoritmi in esso implementati risultano datati e la flessibilità intrinseca di questo codice, potrebbe risultare limitante per l'efficienza di soluzione nel suo complesso. Si è quindi scelto di indagare, utilizzando un codice basato su un algoritmo più recente, il possibile speedup della procedura derivante da una sostituzione del solutore attuale. É importante sottolineare che sebbene il codice utilizzato prometta un incremento nell'efficienza di soluzione esso non rappresenti un candidato per la sostituzione di OpenFOAM poiché, contemporaneamente a questo lavoro, colleghi del dipartimento di scienze e tecnologie aerospaziali, si occupano dello sviluppo di un codice più moderno e con prospettive di efficienza superiori. In questo senso il lavoro di tesi rappresenta un'indagine preliminare sull'effettiva convenienza nell'investire parte delle risorse del gruppo di lavoro nella modifica della procedura attuale.

1.2 Breve descrizione del lavoro

Il codice utilizzato in questo lavoro come confronto con OpenFOAM è bastato sulla combinazione di due elementi fondamentali: un solutore alle differenze finite basato sul dimensional splitting, una tecnica che consiste nello scomporre un problema multidimensionale in una serie di equazioni monodimensionali la cui soluzione numerica risulta più efficiente, e un'implementazione del metodo dei contorni immersi, che consente di introdurre gli effetti di geometrie sulla corrente modificando opportunamente le equazioni di moto. Il solutore è bastato su un algoritmo a correzione di pressione modificato secondo l'idea di Jean-Luc Guermond **3GM2011** che oltre a risolvere l'equazione della quantità di moto utilizzando lo splitting dimensionale già introdotto da Douglass **3D1962**, introduce un nuovo operatore per la soluzione dell'equazione della pressione sostituendo la tipica equazione di Poisson in una serie di tre equazioni monodimensionali da risolvere in cascata. Tale idea influisce su due aspetti cruciali: il primo è che il problema fluidodinamico così posto risulta costituito da soli sistemi lineari tridiagonali che possono essere risolti in modo molto efficiente, il secondo è la parallelizzazione del codice che risulta di più facile implementazione rispetto alla tradizionale equazione di Poisson della pressione. Il codice utilizzato presenta uno scaling lineare fino a 1000 cores **3GM2012** su architetture progettate per questo tipo di calcoli ma promette di scalare altrettanto bene su macchine anche più grandi sulle quali la letteratura presente su OpenFOAM dichiara un crollo nell'efficienza di parallelizzazione **4S2014**.
Il limite principale dell'algoritmo di soluzione del professor Guermond per applicazioni di interesse pratico è la necessità di avere una griglia strutturata parallelepipeda che, in linea di principio non permette la soluzione di casi con geometrie tipiche dei problemi fluidodinamici. A questo proposito l'applicazione del metodo dei contorni immersi rappresenta una soluzione poiché, utilizzando un'implementazione sufficientemente robusta e flessibile come quella proposta da De Tullio **3dTP2016**, è possibile includere gli effetti di geometrie arbitrariamente complesse, anche in movimento, mantenendo la griglia di calcolo cartesiana.
Avvalendosi del codice DNS presentato lo scopo di questo lavoro è indagare quanto sia possibile accelerare l'attuale procedura basata sulla libreria OpenFOAM e, in particolare, sull'algoritmo risolutivo PISO proposto per la prima volta da Issa nel 1986 **4IR1986**, utilizzando come caso di confronto un problema di interesse per il gruppo di ricerca. É importante notare che il caso analizzato è rappresentativo, in quanto a complessità geometrica, della maggior parte dei problemi di interesse pratico a Reynolds $\simeq$ 1000 e

pertanto i risultati ottenuti possono essere estesi a un grande numero di problemi di interesse pratico specialmente in campo medico.

1.3 Struttura della tesi

La tesi è strutturata nel modo seguente:

- *Introduzione*: una breve presentazione del contesto in cui si inserisce questo lavoro e dei suoi obiettivi.
- *Cenni anatomici*: vengono presentate le nozioni fondamentali di anatomia utili per facilitare la comprensione del caso analizzato in termini di geometria, risultati e condizioni da imporre per effettuare le simulazioni .
- *Codice DNS: cenni sul funzionamento*: viene presentato il codice utilizzato in questo lavoro come test per indagare il possibile speedup della procedura. Vengono prima presentate le caratteristiche principali dell'algoritmo di soluzione implementato, in seguito alcuni dettagli sulla trattazione della geometria complessa da simulare e infine alcuni dettagli sull'implementazione parallela del codice.
- *OpenFOAM: cenni sul funzionamento*: viene presentato nelle sue caratteristiche importanti per lo svolgimento di questo lavoro OpenFOAM, lo strumento di consolidato utilizzo nell'attuale procedura del gruppo di ricerca. Prima è fornita una descrizione degli algoritmi di soluzione per le equazioni incomprimibili di Navier-Stokes implementati e successivamente ci si concentra sui solutori disponibili, sulle loro caratteristiche e limiti. Infine sono forniti alcuni cenni sulla parallelizzazione dei solutori utilizzabili.
- *Set-Up simulazioni*: viene presentato il test case scelto per il confronto dei due codici e le motivazioni che hanno condotto a tale scelta. Successivamente per entrambi i software sono descritti gli step necessari per eseguire la simulazione effettuata evidenziando le rispettive criticità.
- *Presentazione dei risultati ottenuti*: Vengono presentati i risultati ottenuti con i due codici e il confronto di prestazioni in termini di tempo di calcolo sia seriale che parallelo
- *Conclusioni e possibili sviluppi futuri*: Nel capitolo conclusivo vengono presentati i limiti dei risultati ottenuti e date indicazioni sul possibile sviluppo futuro del lavoro.
- *Appendice A*: Vengono brevemente riassunti tutti i test preliminari con le rispettive problematiche riscontrate nell'utilizzo del codice utilizzato che hanno portato alla scelta del test case presentato nel capitolo 5.
- *Appendice B*: Vengono dati i dettagli dello sviluppo di una funzione per il codice utilizzato che permetta di esportare i risultati in formato .pvts per accelerare le operazioni di postprocess.

CAPITOLO 2

Cenni anatomici

In questo capitolo è presentata l'anatomia delle vie aeree superiori con lo scopo di permettere al lettore di inquadrare la geometria che definisce il problema fluidodinamico di interesse. Si noti tuttavia che gli aspetti anatomici e fisiologici del problema che il gruppo di lavoro affronta sono di scarsa importanza rispetto agli obiettivi e ai risultati di questo specifico lavoro. Una comprensione di base è comunque utile per poter interpretare più semplicemente i campi di moto presentati e per comprendere le condizioni imposte al fluido nelle simulazioni. Per maggiori dettagli anatomici è possibile consultare testi di anatomia tra i quali quello di riferimento è sicuramente **2Gray2009**.

2.1 Il sistema respiratorio

Il sistema respiratorio umano mostrato schematicamente in figura 2.1 è composto da da una moltitudine di elementi ognuno con determinate funzioni non per forza limitate alla sola respirazione. Una prima divisione macroscopica è quella tra vie aeree superiori e vie aeree inferiori.
Le prime comprendono le cavità anatomiche che permettono, tra l'altro, all'aria esterna di raggiungere i polmoni e sono a loro volta composte da alcuni elementi principali:

- naso esterno
- cavità nasale (o fossa nasale)
- faringe
- laringe alta (prima delle corde vocali)

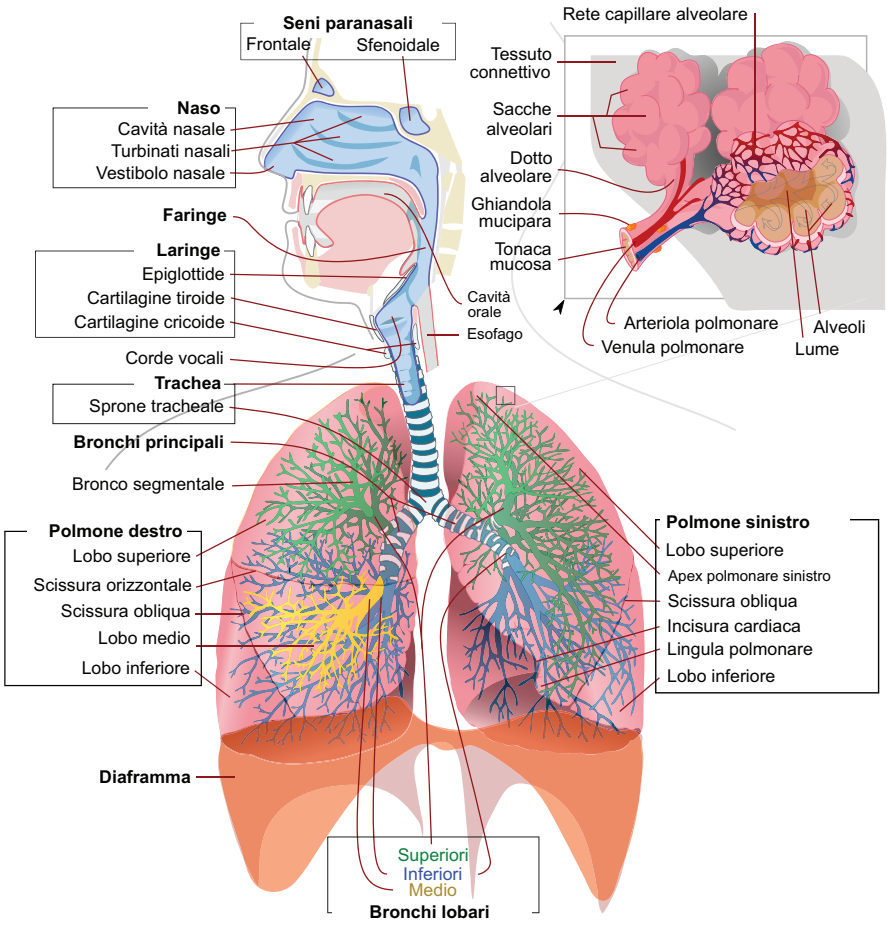


Figura 2.1: *Schematizzazione apparato respiratorio umano*

Le vie aeree inferiori sono invece costituite dagli elementi che si occupano operativamente dell'ematosi, ovvero dello scambio di gas, ossigeno ed anidride carbonica con l'emoglobina. Queste sono costituite da:

- laringe bassa (dopo le corde vocali)
- trachea
- bronchi e bronchioli
- polmoni costituiti da alveoli polmonari
- pleura (costituito da pleura interna ed esterna, al cui interno è presente il liquido pleurico)

Per il contesto in cui si inserisce questo lavoro la geometria fondamentale che si vuole indagare è rappresentata dalle vie aeree superiori, con particolare attenzione a naso esterno, cavità nasale e fossa nasale mentre faringe, laringe e trachea sono inserite nella geometria per una corretta imposizione delle condizioni al contorno. Per questo motivo la descrizione anatomica della vie aeree superiori sarà trattata in modo approfondito mentre sul funzionamento della parte inferiore verranno presentati solo gli elementi

necessari per comprendere il problema fisico analizzato e l'entità delle grandezze in gioco che è necessario imporre per le simulazioni.

2.1.1 Anatomia delle vie aeree superiori

Il naso è la prima struttura delle vie aeree superiori e anatomicamente è formato da una porzione esterna a forma di piramide, situata al centro del volto, e una camera interna in comunicazione con l'ambiente attraverso le narici che convogliano il flusso prima all'interno del vestibolo nasale e poi nelle fosse nasali. Lo scheletro del naso è formato da strutture ossee e cartilaginee: all'interno del massiccio facciale vi è un'apertura piriforme delimitata dalle ossa mascellari e nasali, sulla quale si inseriscono le cartilagini alari, triangolari e quadrangolari che danno forma e supporto al naso.

Internamente il naso presenta la cavità nasale, uno spazio irregolare delimitato anteriormente dal vestibolo nasale, superiormente della base cranica, posteriormente dalle coane che rappresentano il punto di comunicazione con il rinofaringe e inferiormente dalla volta della cavità orale; il setto nasale, formato da una porzione ossea — vomere e lamina perpendicolare dell'etmoide — e una cartilaginea più anteriore, suddivide la cavità in due fosse, pari e simmetriche, delle quali costituisce la parete mediale. La parete laterale (Figura 2.2) ha una struttura più complessa: partecipano alla sua formazione il processo frontale dell'osso mascellare, l'osso nasale, l'osso lacrimale, l'etmoide e la lamina perpendicolare dell'osso palatino; al suo interno presenta tre laminette ossee — i turbinati — che piegano infero-medialmente delimitando i meati:

- Il turbinato inferiore è costituito da una laminetta ossea indipendente che si articola con l'osso mascellare e la lamina perpendicolare dell'osso palatino, e lo spazio così circoscritto rappresenta il meato inferiore, all'interno del quale sbocca il dotto naso-lacrimale.

- Il turbinato medio è un processo dell'osso etmoidale e definisce i confini del meato medio, a questo livello nella parete laterale del naso sono presenti la bulla etmoidale e lo iato semilunare, formato dalla porzione posteriore del processo uncinato e costituente il limite mediale dell'infundibolo etmoidale; il meato medio è la sede di sbocco del canale naso-frontale, dell'ostio del seno mascellare e delle cellette etmoidali anteriori.

- Il turbinato superiore è un processo dell'osso etmoidale ed è il più piccolo e profondo dei cornetti, esso sovrasta il meato superiore al cui interno si apre l'ostio del seno sfenoidale e delle cellette etmoidali del gruppo posteriore.

A livello dei meati si possono individuare due aree funzionali nelle quali si concentrano gli osti dei seni paranasali: il complesso ostio-meatale (COM) nel meato medio e il recesso sfeno-etmoidale (RSE) nel meato superiore.

Il rivestimento mucoso della cavità nasale è formato da un epitelio pavimentoso cheratinizzato nel vestibolo che si trasforma in un epitelio non cheratinizzato per poi diventare un epitelio di tipo respiratorio, pseudostratificato e ciliato: tale mucosa riveste la maggior parte delle cavità nasali ad eccezione della volta del naso dove è presente un epitelio di tipo olfattorio. La lamina propria della mucosa respiratoria contiene numerose ghiandole siero-mucose responsabili delle secrezioni che stratificandosi sulla mucosa intrappolano le particelle presenti nell'aria inalata, inoltre esse contengono sostanze battericide quali lisozima e lattoferrina. Il film mucoso è continuamente fatto avanzare

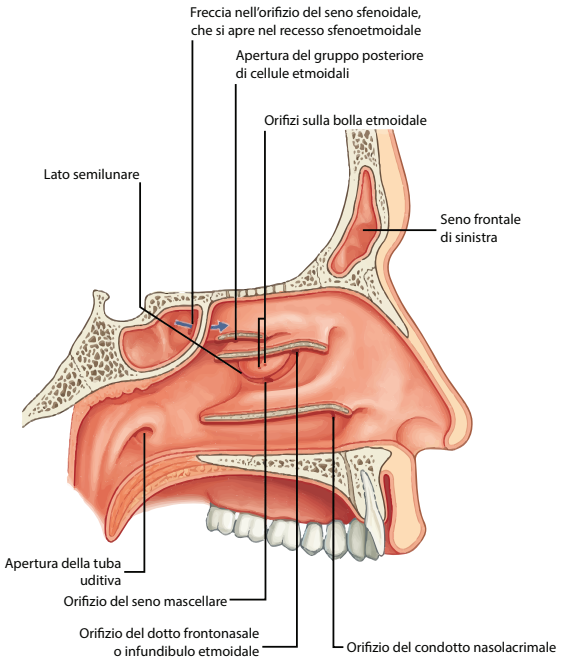


Figura 2.2: *Anatomia della parete laterale del naso (Anatomia del Gray 40° edizione).*

verso il rinofaringe dall'azione delle cellule ciliate e le secrezioni vengono inghiottite. La mucosa olfattoria ricopre 5 cm^2 della porzione supero-posteriore della parete laterale del naso e contiene neuroni olfattivi dispersi tra cellule di sostegno, cellule basali, ghiandole olfattive di Bowman e assoni delle cellule neuronali che decorrono verso la lamina cribra, oltrepassata la quale danno origine al bulbo olfattivo.

I seni paranasali (Figura 2.3) sono strutture pneumatizzate contenute all'interno delle ossa del cranio, comprendono i seni mascellari, frontali, etmoidali e sfenoidali. Sono rivestiti di epitelio respiratorio e comunicano con le fosse nasali attraverso la presenza di osti situati nella parete laterale del naso che consentono la ventilazione delle cavità e il drenaggio delle secrezioni. I seni mascellari sono di forma piramidale con la base rivolta medialmente che forma la maggior parte della parete laterale della cavità nasale e la parete superiore rappresenta una porzione cospicua del pavimento dell'orbita; il loro ostio si apre nell'infundibolo etmoidale, e uno o più osti accessori sono rinvenuti nel $2 \div 44\%$ dei soggetti — a seconda degli studi, infatti la prevalenza è maggiore in quelli eseguiti su cadaveri — **2Hood2009**. Le conseguenze funzionali e patologiche della presenza di osti accessori non sono chiare: non è noto quale sia il rapporto causa-effetto tra la patologia sinusale e il riscontro di osti sovrannumerari; né quanto i cambiamenti di ventilazione possano modificare le fisiologiche caratteristiche del seno. Il completo ricambio di aria all'interno della cavità mascellare è particolarmente lento ed è stato ipotizzato che tale lentezza consenta di minimizzare l'ingresso di patogeni all'interno del seno, nonché permetta di mantenere elevati livelli di NO nell'aria ivi contenuta, grazie anche alla supplementare espressione di enzimi NO-sintetasi, e di contribuire all'idratazione della mucosa **2Hood2009**; la presenza di un ostio accessorio sembra aumentare la ventilazione all'interno della cavità, ma ciò potrebbe provocare il sovvertimento delle proprietà del seno mascellare **2Hood2009**,**2Na2012**. I seni frontali sono situati superiormente

alle arcate sopracciliari, hanno forma triangolare e la loro cavità si apre nell'infundibolo etmoidale. I seni sfenoidali hanno forma irregolare e sono situati posteriormente alla porzione superiore della cavità nasale, il loro ostio è a livello del recesso sfeno-etmoidale, essi sono in rapporto con chiasma ottico e ipofisi superiormente e con carotide interna e seno cavernoso lateralmente. I seni etmoidali sono formati da numerose cellette delimitate da sottili lamine ossee, si localizzano nella porzione superiore della cavità nasale, medialmente all'orbita dalla quale sono separati attraverso la lamina papiracea, clinicamente si riconosce un gruppo anteriore e uno posteriore separati dalla lamina basale del turbinato medio **2Gray2009**,**2Pignataro2012**.

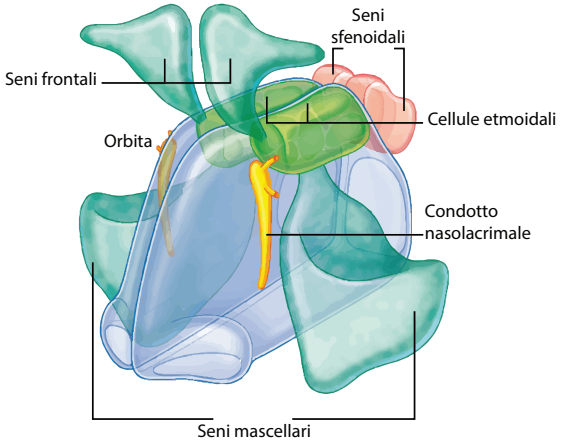


Figura 2.3: *Rappresentazione dei seni paranasali (Anatomia del Gray 40° edizione).*

2.1.2 Cenni anatomici delle vie aeree inferiori

Il polmone è l'organo essenziale per la respirazione, la sua principale funzione è di trasportare l'ossigeno dall'atmosfera al sangue e di espellere l'anidride carbonica dal sangue verso l'ambiente esterno. I polmoni possono funzionare indipendentemente l'uno dall'altro, sia per il nutrimento che per la vascolarizzazione. Sono rivestiti da una membrana chiamata pleura viscerale, che a sua volta si continua in una pleura parietale che riveste la cavità toracica. Il sottile spazio tra le due membrane, spazio pleurico, è ripieno di un liquido che riduce l'attrito tra polmone e parete toracica e aiuta a creare una pressione negativa tra le due membrane, impedendo il collasso dei polmoni e la chiusura delle vie aeree inferiori. La trachea si biforca in due bronchi, ciascuno dei quali conduce ad un polmone. All'interno dei polmoni, i bronchi si ramificano ripetutamente in tubi sempre più sottili chiamati bronchioli, i quali terminano in grappoli di sacche aeree chiamate alveoli. Quest'ultimi cedono al sangue l'ossigeno appena inalato, scambiandolo con l'anidride carbonica, che lo stesso sangue ha trasportato da tutto l'organismo.

Un atto respiratorio è fondamentalmente costituito dalla fase inspiratoria (durata: 1.3 - 1.5 secondi), dalla fase espiratoria (durata: 2.5 – 3 secondi) e da una pausa della durata di circa 0,5 secondi fra le due fasi. L'inspirazione avviene grazie alla contrazione dei muscoli intercostali e del diaframma, che provoca un aumento di volume polmonare e una diminuzione della pressione intrapleurica: ne consegue un'aspirazione dell'aria nei polmoni. L'espirazione solitamente è passiva, determinata dal rilascio della forza

elastica del parenchima polmonare. Il volume toracico diminuisce, i polmoni vengono compressi e l'aria espulsa. La condizione utilizzata per questo lavoro è l'ispirazione normale di un uomo adulto che, statisticamente corrisponde a un volume introdotto nei polmoni di circa 0.5 litri di aria in circa che nel tempo indicato sopra e con una sezione di trachea tipica di circa $2.5 \div 3\, cm^2$ conduce a una velocità media, in condizioni di riposo, di circa $1.1 \div 1.5\, m/s$ in trachea che è un risultato che trova riscontro in letteratura **2EJLJ2013**.

CAPITOLO 3

Codice DNS: cenni sul funzionamento

In questo capitolo vengono introdotti i concetti fondamentali alla base dei metodi di proiezione per la soluzione delle equazioni di Navier-Stokes incomprimibili. Tale classe di metodi, oggi sempre più diffusa, comprende sia il solutore di OpenFOAM che l'algoritmo utilizzato per il confronto. Dopo un' introduzione generale di questi metodi si approfondiranno gli aspetti fondamentali del codice DNS: algoritmo di soluzione, parallelizzazione e implementazione del metodo dei contorni immersi.

3.1 Generalità sui metodi di proiezione

L'algoritmo utilizzato fa parte della classe di metodi numerici di soluzione delle equazioni di Navier-Stokes incomprimibili denominati metodi di proiezione e, più in particolare a correzione di pressione. Questa categoria, introdotta dal lavoro di Chorin e Temam **3C1968** nel 1960, si basa sull'idea di disaccoppiare le equazioni di massa e quantità di moto per risolvere efficientemente il problema complessivo. La semplificazione ottenuta risiede nel fatto che a ogni passo temporale è necessario risolvere una sequenza di equazioni ellittiche disaccoppiate evitando così la complessità, dominante per i problemi tipici della fluidodinamica, di risolvere un sistema di due equazioni accoppiate. Grazie a questo vantaggio i metodi di proiezione rappresentano oggi la scelta più comune per la soluzione delle equazioni incomprimibili di Navier-Stokes.

I metodi a correzione di pressione sono schemi in cui ogni passo di avanzamento temporale è suddiviso logicamente in due sotto-passi: il primo consiste nel calcolare il campo di velocità dalla sola equazione della quantità di moto mentre il secondo nel correggere il campo ottenuto proiettandolo in uno spazio a divergenza nulla in modo da soddisfare il vincolo di incomprimibilità imposto dal principio di conservazione della massa. Da questo secondo passaggio deriva il nome metodi di proiezione con cui si

indica questa tipologia di algoritmi.
A causa delle modifiche migliorative introdotte negli anni, l'algoritmo presentato e utilizzato in questo lavoro non è direttamente riconducibile all'idea presentata; si ritiene pertanto utile riportare i passaggi fondamentali dell'evoluzione dei metodi a correzione di pressione.

Dato lo scopo introduttivo di questa sezione si è scelto per semplicità di notazione e chiarezza espositiva di trascurare il termine non lineare delle equazioni considerando il problema tempo variante di Stokes:

$$\begin{cases} \partial_t \boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + \boldsymbol{\nabla} p = \boldsymbol{f} & \text{in } \Omega \times (0,\, T]\,, \\ \boldsymbol{\nabla} \cdot \boldsymbol{u} = 0 & \text{in } \Omega \times (0,\, T]\,, \\ \boldsymbol{u}|_{\partial\Omega} = \boldsymbol{a} & \text{in } \partial\Omega \times (0,\, T]\,, \\ \boldsymbol{u}|_{t=0} = \boldsymbol{u}_0 & \text{in } \Omega. \end{cases} \tag{3.1}$$

Dove $\boldsymbol{f}$ è un campo forzante regolare e $\boldsymbol{u}_0$ è il campo di velocità iniziale. Se la soluzione fisica è sufficientemente regolare, i risultati presentati possono essere estesi al problema completo di Navier-Stokes poiché il termine trascurato non ha impatto sulle proprietà di convergenza o sull'errore di splitting dovuto alla soluzione disaccoppiata delle equazioni. Tuttavia , adottando questa semplificazione, è necessario tenere presente che, in presenza di condizioni al contorno di Neumann, il termine non lineare ha un'elevata influenza nel determinare se il problema è ben posto o meno **3HRT1996** .

3.1.1 Il metodo di proiezione di Chorin e Temam

Il più semplice metodo di proiezione per risolvere il problema 3.1 è quello proposto da Chorin e Temam:

$$\frac{1}{\Delta t}\left(\tilde{\boldsymbol{u}}^{k+1} - \boldsymbol{u}^{k}\right) - \nu \nabla^2 \tilde{\boldsymbol{u}}^{k+1} = \boldsymbol{f}\left(t^{k+1}\right), \qquad \tilde{\boldsymbol{u}}^{k+1}\Big|_{\Gamma} = 0 \tag{3.2}$$

$$\begin{cases} \dfrac{1}{\Delta t}\left(\boldsymbol{u}^{k+1} - \tilde{\boldsymbol{u}}^{k+1}\right) + \boldsymbol{\nabla} p^{k+1} = 0 \\ \boldsymbol{\nabla} \cdot \boldsymbol{u}^{k+1} = 0, \qquad \boldsymbol{u}^{k+1} \cdot \boldsymbol{n}|_{\Gamma} = 0. \end{cases} \tag{3.3}$$

Nel quale il primo sotto-step 3.2 tiene conto degli effetti viscosi e delle forze esterne mentre il secondo impone la divergenza nulla del campo di velocità. In questa formulazione è evidente l'aggiornamento del campo $\tilde{\boldsymbol{u}}^{k+1}$ al campo a divergenza nulla $\boldsymbol{u}^{k+1}$. Il secondo step è chiamato proiezione poiché definendo l'operatore di Leray:

$$\mathcal{P}\left(\boldsymbol{u}\right) = \boldsymbol{u} - \nabla \Delta^{-1}\left(\boldsymbol{\nabla} \cdot \boldsymbol{u}\right), \tag{3.4}$$

può essere scritto come:

$$\boldsymbol{u}^{k+1} = \mathcal{P}\left(\tilde{\boldsymbol{u}}^{k+1}\right). \tag{3.5}$$

Il metodo presentato risulta avere un errore di splitting del primo ordine che ne impedisce una convergenza più rapida anche utilizzando schemi di discretizzazione temporale più accurati del metodo di Eulero implicito utilizzato nelle equazioni 3.2 e 3.3.

3.1.2 Il metodo di proiezione standard

Goda **3G1979** evidenziò che la scarsa accuratezza del metodo era dovuta all'assenza del contributo della pressione nell'equazione 3.2 e propose di introdurlo attraverso il gradiente del campo di pressione al tempo precedente. Lo schema 3.7 così ottenuto presenta un errore di splitting del secondo ordine ed è oggi noto in letteratura come metodo a correzione di pressione standard:

$$\frac{1}{2\Delta t}\left(3\tilde{\boldsymbol{u}}^{k+1}-4\boldsymbol{u}^{k}+\boldsymbol{u}^{k-1}\right)-\nu\nabla^2\tilde{\boldsymbol{u}}^{k+1}+\boldsymbol{\nabla}p^{k}=\boldsymbol{f}\left(t^{k+1}\right),\qquad \tilde{\boldsymbol{u}}^{k+1}\Big|_{\Gamma}=0 \tag{3.6}$$

$$\begin{cases}\dfrac{1}{2\Delta t}\left(3\boldsymbol{u}^{k+1}-3\tilde{\boldsymbol{u}}^{k+1}\right)+\boldsymbol{\nabla}\left(p^{k+1}-p^{k}\right)=0 & \\ \boldsymbol{\nabla}\cdot\boldsymbol{u}^{k+1}=0, & \boldsymbol{u}^{k+1}\cdot\boldsymbol{n}|_{\Gamma}=0.\end{cases} \tag{3.7}$$

Si noti che lo schema di discretizzazione temporale utilizzato, in questo caso differenze finite indietro del secondo ordine, non influisce sui risultati di convergenza ottenuti purché risulti almeno dello stesso ordine dell'errore di splitting.
Il metodo nella sua forma standard è del secondo ordine per la velocità in norma L^2 ma risulta affetto da un problema numerico che mantiene l'accuratezza al primo ordine sulla velocità in norma H^1 e sulla pressione in norma L^2. Infatti dalle 3.7, poiché per le condizioni al contorno la velocità normale sui bordi del dominio deve essere nulla, si ha:

$$\boldsymbol{\nabla}\left(p^{k+1}-p^{k}\right)\cdot\boldsymbol{n}|_{\Gamma}=0, \tag{3.8}$$

che implica:

$$\boldsymbol{\nabla}p^{k+1}\cdot\boldsymbol{n}|_{\Gamma}=\boldsymbol{\nabla}p^{k}\cdot\boldsymbol{n}|_{\Gamma}=\cdots=\boldsymbol{\nabla}p^{0}\cdot\boldsymbol{n}|_{\Gamma}\,. \tag{3.9}$$

La 3.8 è una condizione sulla pressione non fisica ma introdotta dal metodo numerico che, verificandosi sul contorno del dominio, si manifesta nella soluzione come uno strato limite artificiale.

3.1.3 La forma rotazionale

Per superare il problema della formulazione standard è stata proposta da Timmermans, Minev e Van De Vosse in **3TMV1996** una modifica nel sotto-passo di proiezione:

$$\frac{1}{2\Delta t}\left(3\tilde{\boldsymbol{u}}^{k+1}-4\boldsymbol{u}^{k}+\boldsymbol{u}^{k-1}\right)-\nu\nabla^2\tilde{\boldsymbol{u}}^{k+1}+\boldsymbol{\nabla}p^{k}=\boldsymbol{f}\left(t^{k+1}\right),\quad \tilde{\boldsymbol{u}}^{k+1}\Big|_{\Gamma}=0 \tag{3.10}$$

$$\begin{cases}\dfrac{1}{2\Delta t}\left(3\boldsymbol{u}^{k+1}-3\tilde{\boldsymbol{u}}^{k+1}\right)+\boldsymbol{\nabla}\phi^{k+1}=0 & \\ \boldsymbol{\nabla}\cdot\boldsymbol{u}^{k+1}=0, & \boldsymbol{u}^{k+1}\cdot\boldsymbol{n}|_{\Gamma}=0\end{cases} \tag{3.11}$$

$$\phi^{k+1}=p^{k+1}-p^{k}+\nu\boldsymbol{\nabla}\cdot\tilde{\boldsymbol{u}}^{k+1}. \tag{3.12}$$

Questo schema risulta migliore del precedente, infatti sostituendo la 3.12 nella prima delle 3.11 si ottiene:

$$\frac{1}{2\Delta t}\left(3\boldsymbol{u}^{k+1}-3\tilde{\boldsymbol{u}}^{k+1}\right)+\boldsymbol{\nabla}p^{k+1}-\boldsymbol{\nabla}p^{k}+\nu\boldsymbol{\nabla}\left(\boldsymbol{\nabla}\cdot\tilde{\boldsymbol{u}}^{k+1}\right)=0, \tag{3.13}$$

che utilizzando l'identità vettoriale: $\boldsymbol{\nabla}\left(\boldsymbol{\nabla}\cdot\boldsymbol{u}\right)=\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\boldsymbol{u}+\nabla^2\boldsymbol{u}$ può essere scritta come:

$$\frac{1}{2\Delta t}\left(3\boldsymbol{u}^{k+1}-3\tilde{\boldsymbol{u}}^{k+1}\right)+\boldsymbol{\nabla}p^{k+1}-\boldsymbol{\nabla}p^{k}+\nu\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\tilde{\boldsymbol{u}}^{k+1}+\nu\nabla^2\tilde{\boldsymbol{u}}^{k+1}=0. \tag{3.14}$$

Sommando questa espressione con la 3.10 si ottiene:

$$\frac{1}{2\Delta t}\left(3\boldsymbol{u}^{k+1}-4\boldsymbol{u}^{k}+\boldsymbol{u}^{k-1}\right)+\nu\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\tilde{\boldsymbol{u}}^{k+1}+\boldsymbol{\nabla}p^{k+1}=\boldsymbol{f}\left(t^{k+1}\right). \tag{3.15}$$

Poiché però il rotore del gradiente di un campo è identicamente nullo, applicando l'operatore di doppio rotore alla prima delle 3.11 si ottiene

$$\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\tilde{\boldsymbol{u}}^{k+1}=\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\boldsymbol{u}^{k+1}. \tag{3.16}$$

La 3.15 si può quindi riscrivere, insieme alla condizione di divergenza nulla come:

$$\begin{cases}\dfrac{1}{2\Delta t}\left(3\boldsymbol{u}^{k+1}-4\boldsymbol{u}^{k}+\boldsymbol{u}^{k-1}\right)+\nu\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\boldsymbol{u}^{k+1}+\boldsymbol{\nabla}p^{k+1}=\boldsymbol{f}\left(t^{k+1}\right)\\ \boldsymbol{\nabla}\cdot\boldsymbol{u}^{k+1}=0, \qquad\qquad \boldsymbol{u}^{k+1}\cdot\boldsymbol{n}|_{\Gamma}=0.\end{cases} \tag{3.17}$$

Valutando la prima equazione sul contorno del dominio lungo la normale, la velocità si annulla per le condizioni al contorno e rimane:

$$\partial_n p^{k+1}\Big|_{\Gamma}=\left(\boldsymbol{f}\left(t^{k+1}\right)-\nu\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\boldsymbol{u}^{k+1}\right)\cdot\boldsymbol{n}|_{\Gamma}, \tag{3.18}$$

che, al contrario della 3.8 è una condizione consistente per la pressione. L'errore di splitting si manifesta in questo caso solamente come una errata condizione al contorno per la velocità tangente. Utilizzando schemi di discretizzazione sufficientemente accurati il metodo a correzione di pressione in forma rotazionale risulta di secondo ordine per la velocità in norma L^2 mentre di ordine 1.5 per pressione in norma L^2 e velocità in norma H^1.

3.1.4 Generica forma del metodo di proiezione

Diverse formulazioni numericamente stabili del metodo a correzione di pressione possono essere ottenute introducendo il coefficiente $0\leq\chi\leq 1$ che se uguale a zero conduce alla formulazione standard del metodo mentre se uguale a 1 a quella rotazionale. La forma generica assume quindi la forma:

$$\frac{1}{2\Delta t}\left(3\tilde{\boldsymbol{u}}^{k+1}-4\boldsymbol{u}^{k}+\boldsymbol{u}^{k-1}\right)-\nu\nabla^2\tilde{\boldsymbol{u}}^{k+1}+\boldsymbol{\nabla}p^{k}=\boldsymbol{f}\left(t^{k+1}\right), \qquad \tilde{\boldsymbol{u}}^{k+1}\Big|_{\Gamma}=0 \tag{3.19}$$

$$\begin{cases}\dfrac{1}{2\Delta t}\left(3\boldsymbol{u}^{k+1}-3\tilde{\boldsymbol{u}}^{k+1}\right)+\boldsymbol{\nabla}\phi^{k+1}=0\\ \boldsymbol{\nabla}\cdot\boldsymbol{u}^{k+1}=0, \qquad\qquad \boldsymbol{u}^{k+1}\cdot\boldsymbol{n}|_{\Gamma}=0\end{cases} \tag{3.20}$$

$$\phi^{k+1}=p^{k+1}-p^{k}+\chi\nu\boldsymbol{\nabla}\cdot\tilde{\boldsymbol{u}}^{k+1}. \tag{3.21}$$

É opportuno sottolineare che i coefficienti e lo stencil necessario dipendono dagli schemi di discretizzazione utilizzati che, viste le proprietà di convergenza del metodo, è opportuno scegliere del secondo ordine.

3.1.5 Il campo di velocità corretto

Una domanda speso controversa in letteratura è quale campo di velocità tra $\tilde{\boldsymbol{u}}$ e $\boldsymbol{u}$ sia effettivamente corretto, infatti seppur il primo non sia a divergenza nulla soddisfa le condizioni al contorno imposte, mentre il secondo risulta a divergenza nulla ma non

soddisfa le corrette condizioni al contorno. Dal punto di vista dell'accuratezza non c'è differenza nel prendere il primo o il secondo poiché entrambi i campi hanno le stesse proprietà di convergenza; dal punto di vista implementativo, come evidenziato per la prima volta in J. L. Guermond **3G1996**, ci sono due motivi per eliminare il campo di velocità proiettato $\boldsymbol{u}$. Il primo motivo riguarda l'implementazione con gli elementi finiti e non è di interesse per questo lavoro (si veda l'articolo di J. L. Guermond per maggiori dettagli), il secondo è che il campo di velocità proiettato può essere rimosso dall'algoritmo con una semplice manipolazione algebrica riducendo il costo computazionale. Nell'equazione di aggiornamento della velocità 3.19 il campo $\boldsymbol{u}$ compare semplicemente ai passi temporali precedenti e può essere quindi sostituito dal campo $\tilde{\boldsymbol{u}}$ se si decide di utilizzare quest'ultimo come campo di velocità corretto. Nell'equazione di correzione 3.20 $\boldsymbol{u}$ può essere eliminato applicando l'operatore di divergenza, infatti poiché il campo $\boldsymbol{u}$ per definizione soddisfa il vincolo di incomprimibilità scompare e si ottiene un'equazione di Poisson per ϕ con la relativa condizione al contorno:

$$\nabla^2\phi^{k+1} = \frac{3}{2\Delta t}\boldsymbol{\nabla}\cdot\tilde{\boldsymbol{u}}^{k+1}, \qquad \partial_n\phi^{k+1}\Big|_\Gamma = 0. \tag{3.22}$$

In conclusione quindi il metodo di proiezione per il problema di Stokes tempo variante, utilizzando le differenze finite indietro del secondo ordine e eliminando il campo di velocità corretto può essere scritto come:

$$\frac{1}{2\Delta t}\left(3\tilde{\boldsymbol{u}}^{k+1} - 4\boldsymbol{u}^k + \boldsymbol{u}^{k-1}\right) - \nu\nabla^2\tilde{\boldsymbol{u}}^{k+1} + \boldsymbol{\nabla}p^k = \boldsymbol{f}\left(t^{k+1}\right), \qquad \tilde{\boldsymbol{u}}^{k+1}\Big|_\Gamma = 0 \tag{3.23}$$

$$\nabla^2\phi^{k+1} = \frac{3}{2\Delta t}\boldsymbol{\nabla}\cdot\tilde{\boldsymbol{u}}^{k+1}, \qquad \partial_n\phi^{k+1}\Big|_\Gamma = 0 \tag{3.24}$$

$$\phi^{k+1} = p^{k+1} - p^k + \chi\nu\boldsymbol{\nabla}\cdot\tilde{\boldsymbol{u}}^{k+1}. \tag{3.25}$$

In questa forma, che è quella con cui verrà presentato l'algoritmo del codice utilizzato, è possibile riconosce tre passaggi logici ad ogni passo temporale: Previsione della velocità, correzione (senza il calcolo del campo corretto) e aggiornamento della pressione.

3.2 L'algoritmo di soluzione del codice DNS

Il codice utilizzato in questo lavoro è bastato su un solutore a correzione di pressione che implementa l'idea di J. L. Guermond **3GM2012** che dovrebbe garantire un'efficienza superiore rispetto agli algoritmi implementati in OpenFOAM e una migliore scalabilità parallela se confrontato con i metodi, anche spettrali, basati sulla correzione attraverso la soluzione dell'equazione di Poisson per la pressione.

3.2.1 Il metodo di Guermond

Consideriamo le equazioni incomprimibili di Navier-Stokes dipendenti dal tempo nel dominio parallelepipedo $\Omega = [0,\, L_x] \times [0,\, L_y] \times [0,\, L_z] \subset \mathbb{R}^3$:

$$\begin{cases} \partial_t\,\boldsymbol{u} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u} - \dfrac{1}{Re}\nabla^2\boldsymbol{u} + \boldsymbol{\nabla}p = \boldsymbol{f} & \text{in } \Omega\times(0,\,T]\,, \\ \boldsymbol{\nabla}\cdot\boldsymbol{u} = 0 & \text{in } \Omega\times(0,\,T]\,, \\ \boldsymbol{u}|_{\partial\Omega} = \boldsymbol{a} & \text{in } \partial\Omega\times(0,\,T]\,, \\ \boldsymbol{u}|_{t=0} = \boldsymbol{u}_0 & \text{in } \Omega. \end{cases} \tag{3.26}$$

Dove $\boldsymbol{f}$ rappresenta un termine di sorgente, $\boldsymbol{a}$ le condizioni al contorno e $\boldsymbol{u}_0$ la condizione iniziale. Come già accennato nella sezione precedente, virtualmente tutti gli schemi conosciuti basati sul metodo a correzione di pressione risolvono la seguente perturbazione singolare della versione linearizzata del sistema 3.26:

$$\begin{cases} \partial_t \boldsymbol{u}_\epsilon - \dfrac{1}{Re}\nabla^2 \boldsymbol{u}_\epsilon + \boldsymbol{\nabla} p_\epsilon = \boldsymbol{f} & \text{in } \Omega \times (0,\, T]\,, \quad \boldsymbol{u}_\epsilon|_{\partial\Omega \times (0,\,T]} = \boldsymbol{a}, \quad \boldsymbol{u}_\epsilon|_{t=0} = 0 \\ -\Delta t\, \nabla^2 \phi_\epsilon + \boldsymbol{\nabla} \cdot \boldsymbol{u}_\epsilon = 0 & \text{in } \Omega \times (0,\, T]\,, \quad \partial_n \phi_\epsilon|_{\partial\Omega \times (0,\,T]} = 0 \\ \Delta t\, \partial_t p_\epsilon = \phi_\epsilon - \dfrac{\chi}{Re}\boldsymbol{\nabla} \cdot \boldsymbol{u}_\epsilon & p_\epsilon|_{t=0} = p_0. \end{cases} \tag{3.27}$$

Dove $p_0 = p|_{t=0}$, Δt è il parametro di perturbazione (ovvero $\epsilon = \Delta t$) e $\chi \in [0,\, 1]$ è un parametro di controllo. É possibile dimostrare le proprietà di convergenza dello schema 3.27 **3GM2011** e, che in particolare, $\boldsymbol{u}_\epsilon$ è una perturbazione $\mathcal{O}\left(\Delta t^2\right)$ in norma $\boldsymbol{L}^2$ e una perturbazione $\mathcal{O}\left(\Delta t^{3/2}\right)$ in norma $\boldsymbol{H}^1$ per ogni $0 < \chi \leq 1$.
La seconda equazione delle 3.27 può essere generalizzata sostituendo il laplaciano della pressione con un generico operatore $A\,[-]$ ottenendo:

$$\begin{cases} \partial_t \boldsymbol{u}_\epsilon - \dfrac{1}{Re}\nabla^2 \boldsymbol{u}_\epsilon + \boldsymbol{\nabla} p_\epsilon = \boldsymbol{f} & \text{in } \Omega \times (0,\, T]\,, \quad \boldsymbol{u}_\epsilon|_{\partial\Omega \times (0,\,T]} = \boldsymbol{a}, \quad \boldsymbol{u}_\epsilon|_{t=0} = 0 \\ \Delta t\, A\phi_\epsilon + \boldsymbol{\nabla} \cdot \boldsymbol{u}_\epsilon = 0 & \text{in } \Omega \times (0,\, T]\,, \quad \phi_\epsilon \in D\,(A) \\ \Delta t\, \partial_t p_\epsilon = \phi_\epsilon - \dfrac{\chi}{Re}\boldsymbol{\nabla} \cdot \boldsymbol{u}_\epsilon & p_\epsilon|_{t=0} = p_0. \end{cases} \tag{3.28}$$

É possibile dimostrare che l'equazione 3.28 ha le stesse proprietà di convergenza della 3.27 se l'operatore A e il suo dominio $D\,(A)$ sono tali che la forma bilineare

$$a\,(p, q) := \int_\Omega qAp\, d\boldsymbol{x}, \tag{3.29}$$

soddisfa le seguenti proprietà:

$$a \text{ è simmetrica} \tag{3.30}$$

$$\|\boldsymbol{\nabla} q\|^2_{\boldsymbol{L}^2} \leq a\,(q, q) \quad \forall q \in D\,(A)\,. \tag{3.31}$$

Sono possibili diverse scelte dell'operatore A, il solutore utilizzato si basa sulla proposta di J.L. Guermond di utilizzare l'operatore:

$$A := (1 - \partial_{xx})\,(1 - \partial_{yy})\,(1 - \partial_{zz})\,, \tag{3.32}$$

che con le corrette condizioni al contorno soddisfa le proprietà 3.31 e 3.32.

L'algoritmo proposto risolve l'equazione della quantità di moto utilizzando lo schema del secondo ordine proposto da Douglas **3D1962** e corregge il campo di pressione utilizzando l'operatore definito da 3.32. In particolare la soluzione di un passo temporale si articola in quattro fasi:

1. *Previsione della pressione:* detto p_0 il campo di pressione al tempo $t = 0$, l'algoritmo è inizializzato imponendo $p^{-\frac{1}{2}} = p_0$ e $\phi^{-\frac{1}{2}} = 0$. Dopo di che per ogni passo temporale $n \geq 0$ la previsione della pressione è calcolata come:

$$p^{*,n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi^{n-\frac{1}{2}}. \tag{3.33}$$

2. *Aggiornamento velocità:* Il campo di velocità è inizializzato imponendo $\boldsymbol{u}^0 = \boldsymbol{u}_0$, e per ogni passo temporale $n \geq 0$ la velocità è aggiornata risolvendo in cascata i seguenti problemi monodimensionali:

$$\begin{aligned}
&\frac{\boldsymbol{\xi}^{n+1} - \boldsymbol{u}^n}{\Delta t} - \frac{1}{Re}\nabla^2 \boldsymbol{u}^n + \boldsymbol{\nabla} p^{*,n+\frac{1}{2}} + \boldsymbol{NL}^{n+1}\left(\boldsymbol{u}^n, \boldsymbol{u}^{n-1}\right) = 0 && \boldsymbol{\xi}^{n+1}\Big|_{\partial\Omega} = \boldsymbol{a},\\
&\frac{\boldsymbol{\eta}^{n+1} - \boldsymbol{\xi}^{n+1}}{\Delta t} - \frac{1}{2Re}\partial_{xx}\left(\boldsymbol{\eta}^{n+1} - \boldsymbol{u}^n\right) = 0 && \boldsymbol{\eta}^{n+1}\Big|_{x=0,\,L_x} = \boldsymbol{a},\\
&\frac{\boldsymbol{\zeta}^{n+1} - \boldsymbol{\eta}^{n+1}}{\Delta t} - \frac{1}{2Re}\partial_{yy}\left(\boldsymbol{\zeta}^{n+1} - \boldsymbol{u}^n\right) = 0 && \boldsymbol{\zeta}^{n+1}\Big|_{y=0,\,L_y} = \boldsymbol{a},\\
&\frac{\boldsymbol{u}^{n+1} - \boldsymbol{\zeta}^{n+1}}{\Delta t} - \frac{1}{2Re}\partial_{zz}\left(\boldsymbol{u}^{n+1} - \boldsymbol{u}^n\right) = 0 && \boldsymbol{u}^{n+1}\Big|_{z=0,\,L_z} = \boldsymbol{a}.
\end{aligned} \tag{3.34}$$

Dove il termine non lineare è calcolato come

$$\boldsymbol{NL}^{n+1}\left(\boldsymbol{u}^n, \boldsymbol{u}^{n-1}\right) = \frac{3}{2}\left(\boldsymbol{u}^n \cdot \boldsymbol{\nabla}\right)\boldsymbol{u}^n - \frac{1}{2}\left(\boldsymbol{u}^{n-1} \cdot \boldsymbol{\nabla}\right)\boldsymbol{u}^{n-1}. \tag{3.35}$$

3. *Penalità:* L'incremento di pressione $\phi^{n+\frac{1}{2}}$ è calcolata da ogni passo temporale $n \geq 0$ risolvendo in cascata i seguenti problemi monodimensionali:

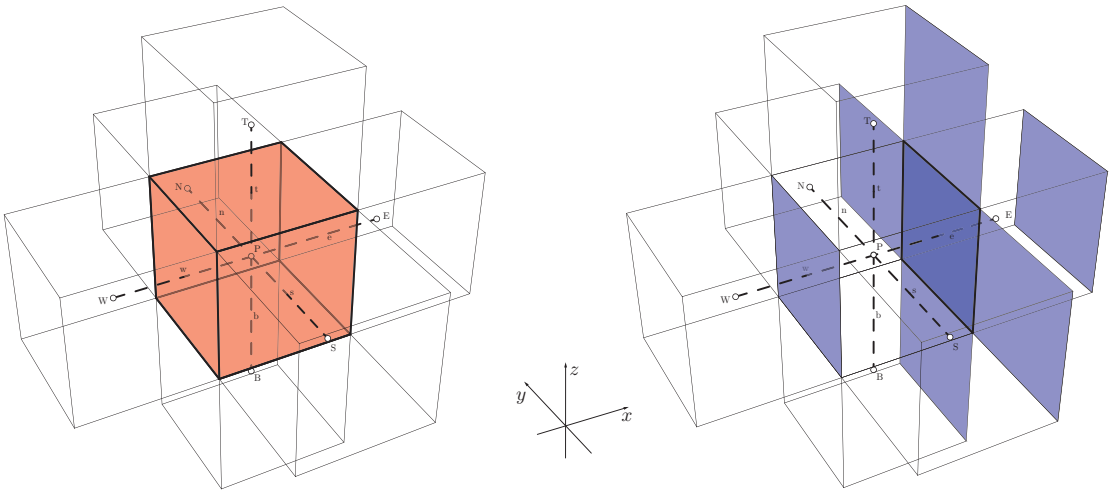
$$\begin{aligned}
&\psi - \partial_{xx}\psi = -\frac{1}{\Delta t}\boldsymbol{\nabla} \cdot \boldsymbol{u}^{n+1} && \partial_x\psi|_{x=0,\,L_x} = 0,\\
&\varphi - \partial_{yy}\varphi = \psi && \partial_y\varphi|_{y=0,\,L_y} = 0,\\
&\phi^{n+\frac{1}{2}} - \partial_{zz}\phi^{n+\frac{1}{2}} = \varphi && \partial_z\phi^{n+\frac{1}{2}}\Big|_{z=0,\,L_z} = 0.
\end{aligned} \tag{3.36}$$

4. *Aggiornamento pressione:* Infine la pressione viene aggiornata per il passo temporale successivo:

$$p^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi^{n+\frac{1}{2}} - \frac{\chi}{Re}\boldsymbol{\nabla} \cdot \frac{1}{2}\left(\boldsymbol{u}^{n+1} + \boldsymbol{u}^n\right). \tag{3.37}$$

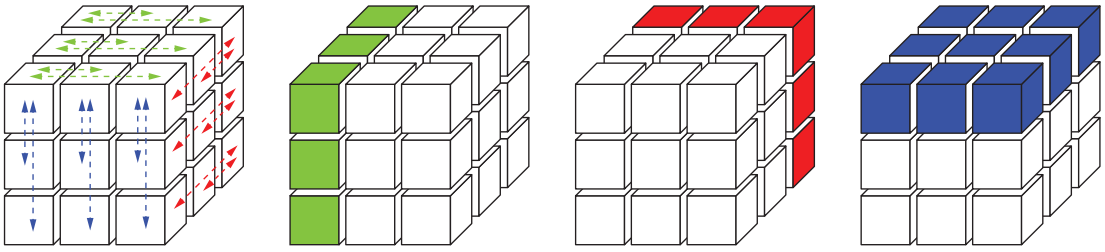
3.2.2 Implementazione parallela

L'algoritmo presentato è implementato con una discretizzazione spaziale basata su uno stencil MAC (Marker-And-Cell)**3MTF2007** utilizzando le differenze finite centrate per le derivate sia di primo che di secondo ordine. La griglia utilizzata, per evitare il noto problema di instabilità del campo di pressione a scacchiera **4HW1965** è sfalsata, ovvero le diverse variabili (u, v, w, p) sono definite e calcolate in punti spazialmente distinti dello spazio. In particolare, con riferimento alla figura (3.1) la pressione è definita al centro di ogni cella mentre le componenti della velocità nel centro delle facce normali alla componente considerata. Su ogni faccia di ciascuna cella è definita solo una componente della velocità (quella ortogonale) e non l'intero vettore.
L'algoritmo è parallelizzato su una decomposizione cartesiana del dominio (come mostrato in figura 3.2) utilizzando la libreria Message Passing Interface (MPI).
Una volta eseguita la discretizzazione spaziale, tutti i problemi monodimensionali presentati nell'algoritmo risultano descritti da sistemi lineari tridiagonali che possono essere risolti in parallelo utilizzando il solutore diretto del complemento di Schur**3HKK2005** derivante dalla decomposizione del dominio. In particolare la procedura si articola in



(a) (b)

Figura 3.1: *(a) Griglia sfalsata in tre dimensioni supponendo per semplicità elementi cubici, la pressione è definita nei punti principali della griglia(P, E, ecc..).(b) Intorno della velocità u_e*



(a) Comunicazioni

(b) Processori incaricati del complemento di Schur lungo x

(c) Processori incaricati del complemento di Schur lungo y

(d) Processori incaricati del complemento di Schur lungo z

Figura 3.2: *Comunicazioni e processori incaricati della soluzione del complemento di Schur*

questo modo: in ogni direzione la partizione del dominio induce la separazione in incognite interne u_i e incognite di interfaccia u_e; come conseguenza è necessario, per ogni istante di tempo e direzione spaziale d con $d \in (1, 2, 3)$, risolvere un grande numero di sistemi lineari del tipo:

$$\begin{pmatrix} A_{ii} & A_{ie} \\ A_{ei} & A_{ee} \end{pmatrix} \begin{pmatrix} u_i \\ u_e \end{pmatrix} = \begin{pmatrix} f_i \\ f_e \end{pmatrix}. \tag{3.38}$$

La soluzione di ogni sistema del tipo 3.38 è costruita risolvendo prima il problema per u_e:

$$\left(A_{ee} - A_{ei}A_{ii}^{-1}A_{ie}\right) u_e = f_e - A_{ei}A_{ii}^{-1}f_i, \tag{3.39}$$

e successivamente ricavando u_i da

$$A_{ii}u_i = f_i - A_{ie}u_e. \tag{3.40}$$

Indicando con n_e il numero di interfacce nella d-esima direzione, il complemento di Schur definito come:

$$S := A_{ee} - A_{ei}A_{ii}^{-1}A_{ie}, \tag{3.41}$$

è una matrice tridiagonale di dimensione $n_e \times n_e$. Tale matrice è costruita una volta sola in fase di pre-processing e memorizzata da uno dei P_d processori incaricati di risolvere la d-esima direzione. Indicando con P il numero totale di processori disponibili e supponendo una decomposizione isotropa del dominio si ha $P_d = P^{1/3}$. Operativamente parlando, l'algoritmo procede prima risolvendo, su ogni processore, il sistema tridiagonale locale

$$A_{ii}x_i = f_i, \tag{3.42}$$

per poi comunicare la quantità $A_{ei}x_i$ al processore dove è memorizzato il complemento di Schur. Su quest'ultimo, grazie alle informazioni di tutti i processori, viene assemblata la quantità

$$f_e - A_{ei}A_{ii}^{-1}f_i, \tag{3.43}$$

e risolto il sistema lineare:

$$Su_e = f_e - A_{ei}A_{ii}^{-1}f_i. \tag{3.44}$$

Infine la quantità $A_{ie}u_e$ è restituita ad ogni processore e, su ciascuno è risolto il sistema lineare

$$A_{ii}u_i = f_i - A_{ie}u_e. \tag{3.45}$$

É importante notare che la soluzione dei sistemi del tipo 3.45 non necessita di nessuna comunicazione tra i vari processori. Pertanto il numero totale di comunicazioni per ogni processore in ogni direzione consiste nell'invio di un vettore di dimensione uguale al numero delle incognite di interfaccia(ogni sotto-dominio dovuto alla decomposizione ha, in ogni direzione, due interfacce) e nel ricevere un vettore di analoga dimensione. Il processo di comunicazione è illustrato in figura 3.2. Il numero di sistemi tridiagonali da risolvere in ogni direzione è elevato, in particolare indicando con N il numero totale di punti nella griglia e assumendo una decomposizione isotropa del dominio ci sono

$$n^2 = \left(\frac{N}{P}\right)^{2/3}, \tag{3.46}$$

sistemi da risolvere per ogni riga di processori. La procedura di soluzione descritta può essere accelerata memorizzando il complemento di Schur in ogni processore nella riga considerata e assegnando ad ogni processore della riga la soluzione di $n^2/P^{1/3}$ sistemi del tipo eq. 3.28 al posto che affidare la soluzione dell'intero sistema a un singolo processore.
Secondo i test eseguiti **3GM2012** la scalabilità del codice descritto è lineare fino ad almeno 1024 processori, l'algoritmo risulta stabile nel regime di Navier-Stokes sotto la condizione CFL e l'algoritmo è accurato al secondo ordine rispetto al tempo considerando la norma L2 della velocità.

Parallelizzazione: Poisson vs direction splitting

Negli algoritmi tradizionali la correzione della pressione è calcolata risolvendo l'equazione di Poisson:

$$-\Delta\phi^{n+1} = -\frac{1}{\Delta t}\nabla \cdot u^{n+1} \qquad \text{con } \partial_n\phi|_{\partial\Omega\times(0,T]} = 0. \tag{3.47}$$

Per problemi di grosse dimensioni e alti numeri di Reynolds, il costo di risolvere tale equazione diventa dominate anche a causa delle difficoltà nel parallelizzare efficacemente la soluzione della 3.47 su un grande numero di processori. Nell'algoritmo descritto la correzione della pressione è calcolata risolvendo la sequenza di problemi monodimensionali:

$$\begin{aligned} &\psi - \partial_{xx}\psi = -\frac{1}{\Delta t}\boldsymbol{\nabla}\cdot\boldsymbol{u}^{n+1} && \partial_x\psi|_{x=0,\,L_x} = 0, \\ &\varphi - \partial_{xx}\varphi = \psi && \partial_y\varphi|_{y=0,\,L_y} = 0, \\ &\phi^{n+\frac{1}{2}} - \partial_{zz}\phi^{n+\frac{1}{2}} = \varphi && \partial_z\phi^{n+\frac{1}{2}}\Big|_{z=0,\,L_z} = 0. \end{aligned} \tag{3.48}$$

Tale procedimento risulta molto più semplice da parallelizzare tanto che, nonostante i metodi per risolvere l'eq. 3.47 basati sull FFT risultino molto efficienti, è possibile dimostrare che, in termini di comunicazioni tra processori, risolvere il sistema 3.48 risulta più efficiente. Per chiarire questa affermazione consideriamo un dominio cubico contenente N punti che vogliono essere risolti mediante l'utilizzo di P processori. Per quanto reperito in letteratura, gli algoritmi basati sulla FFT per risolvere l'equazione di Poisson utilizzano una decomposizione dei dati a codominio unitario. In tre dimensioni, come mostrato in figura 3.3, questo significa che i dati sono divisi in sotto-domini parallelepipedi di lunghezza uguale a quella dell'intero dominio nella direzione considerata, ciascuno contenete $N = P$ punti.
La co-dimensione unitaria nella partizione dei dati è necessaria a causa della natura globale della FFT monodimensionale (per maggiori dettagli sull'FFT e la sua parallelizzazione si vedano, per esempio **3CBGT2008** e **3SR2008**). Una volta risolte le varie FFT ogni processore riutilizza

$$\frac{N/P}{\sqrt{P}} \tag{3.49}$$

dati rispetto agli $N = P$ che contiene mentre i restanti

$$\left(1 - \frac{1}{\sqrt{P}}\right)\frac{N}{P}, \tag{3.50}$$

devono essere inviati agli altri $P^{1/2} - 1$ processori in modo che le FFT monodimensionali possano essere eseguite nelle altre direzioni. Ogni processore deve ricevere in ingresso una quantità analoga di dati e quindi complessivamente invia e riceve

$$2\left(1 - \frac{1}{\sqrt{P}}\right)\frac{N}{P}, \tag{3.51}$$

dati per ogni direzione da risolvere. In tre dimensioni si hanno

$$6\left(1 - \frac{1}{\sqrt{P}}\right)\frac{N}{P}, \tag{3.52}$$

dati scambiati ad ogni passo temporale per ogni processore.
Una maniera più naturale di distribuire i punti quando si vuole risolvere il sistema 3.48 è una griglia, mostrata in figura 3.4, cartesiana cubica composta da $P^{1/3}$ processori in ogni direzione, così che ogni processore contenga n^3 con

$$n = (N/P)^{1/3}\,. \tag{3.53}$$

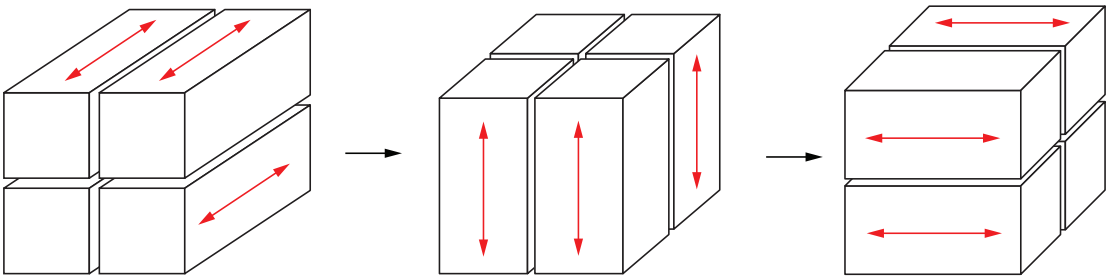


Figura 3.3: *Partizione del dominio per l'implementazione parallela mediante FFT con $P = 4$ processori. Le frecce indicano la direzione lungo la quale è effettuata la trasformata*

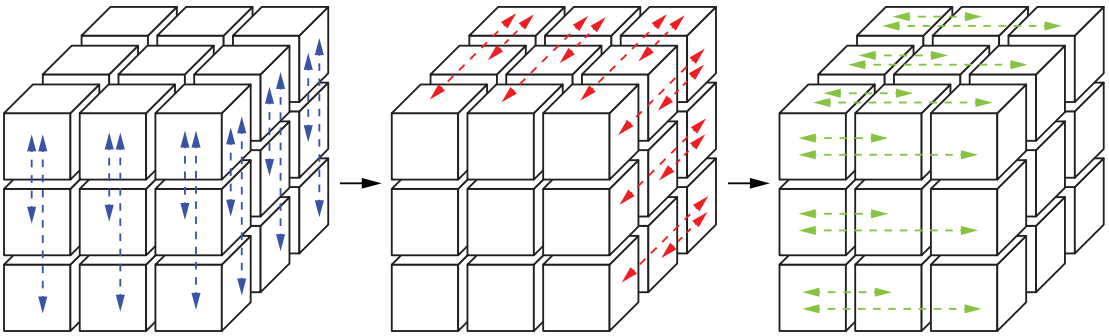


Figura 3.4: *Partizione del dominio per l'implementazione parallela mediante l'algoritmo di splitting direzionale utilizzando la tecnica del complemento di Schur su $P = 27$ processori. Le frecce indicano le comunicazioni che sono effettuate in ogni direzione risolta*

Utilizzando la tecnica del complemento di Schur per risolvere i problemi monodimensionali relativi alla correzione della pressione ogni processore ha bisogno di scambiare solo i dati di interfaccia con i processori che sono incaricati di risolvere il sistema del complemento di Schur. Quindi l'entità totale dei dati che vengono scambiati (inviati e ricevuti) da ogni processore sono inferiori a

$$3 \cdot 4 \cdot n^2 = 12 \left(N/P\right)^{2/3}, \tag{3.54}$$

per processore, ad ogni passo temporale.
In conclusione il numero di comunicazioni tra processori necessarie ad ogni passo temporale utilizzando un algoritmo FFT è significativamente più grande rispetto a quello necessario all'algoritmo basata sullo splitting dimensionale. Considerando per esempio un milione di punti per processore, ovvero $\frac{N}{P} = 10^6$ e volendo utilizzare $P = 1000$ processori, il rapporto tra il numero di comunicazioni necessarie utilizzando un algoritmo FFT ($n_{c_{FFT}}$) e quelle necessarie all'algoritmo presentato ($n_{c_{ds}}$) è:

$$\frac{n_{c_{FFT}}}{n_{c_{ds}}} = \frac{6\left(1 - \frac{1}{\sqrt{P}}\right)\frac{N}{P}}{12\left(\frac{N}{P}\right)^{2/3}} = \frac{1}{2}\left(1 - \frac{1}{\sqrt{P}}\right)\left(\frac{N}{P}\right)^{1/3} \approx 48. \tag{3.55}$$

Ovvero la correzione della pressione eseguita con una tecnica FFT impiega circa cinquanta volte le comunicazioni tra processori impiegate dalla correzione eseguita con la tecnica del complemento di Schur.

3.3 Trattazione della geometria: metodo dei contorni immersi

3.3.1 Introduzione

Recentemente si è assistito a un rinnovato interesse per gli algoritmi, noti come metodi dei contorni immersi, che non necessitino di griglie conformi alla geometria da analizzare. Questi presentano diversi vantaggi rispetto ai metodi basati su griglie conformi non strutturate nella soluzione di problemi con geometrie molto complesse o di interazione fluido-struttura. In questi metodi, infatti, le equazioni di Navier-Stokes vengono risolte su una griglia che non dovendo tener conto della geometria del problema può essere generata con semplicità e non deve essere deformata in seguito a spostamenti della geometria di interesse. Dipendentemente dalla formulazione implementata, gli effetti geometrici sono inseriti nel calcolo modificando opportunamente lo stencil in prossimità del corpo (**3USR1996**) utilizzando forzanti applicate al problema continuo (**3P1972**) prima della discretizzazione o direttamente alle equazioni discretizzate (**3BEP2000**). In quest'ultimo caso il metodo è chiamato delle forzanti dirette, e risulta particolarmente interessante per i calcoli attorno a complessi corpi rigidi dal momento che le forzanti possono essere facilmente introdotte nella struttura di solutori a differenze finite o volumi finiti già esistenti. Per contorni in movimento si presentano ulteriori difficoltà, infatti un'implementazione diretta del metodo porta a forze idrodinamiche che, a causa della scarsa regolarità, possono dare problemi di instabilità al codice (**3U2005**, **3YB2006**). Yang e Balaras **3YB2006** hanno evidenziato come le grandi oscillazioni delle forze idrodinamiche sui contorni immersi in movimento siano dovute ai punti con valori di pressione e velocità errati poiché interni alla geometria negli istanti precedenti. Tra le varie tecniche per affrontare questa problematica presenti in letteratura **3YB2006** **3MDBNVL2008**, si è scelta una variante del metodo di Uhlmann **3U2005** che, per primo ha proposto di calcolare le forzanti su punti Lagrangiani della superficie invece che su quelli Euleriani della griglia e di usare funzioni impulso regolarizzate per passare le variabili da una griglia all'altra. Quest'ultima scelta, seppur risulti robusta e efficiente nel caso di validazione di una particella solida sospesa in una corrente fluida, risulta limitante per un estensione dell'algoritmo a casi più complessi come quello di interesse. Per questo motivo nel codice utilizzato la formulazione è quella presentata da **3VB2009** che rivisita il metodo di Uhlmann utilizzando un'approssimazione ai minimi quadrati per costruire le funzioni di trasferimento per passare dalla griglia Euleriana a quella Lagrangiana e viceversa. Per l'implementazione di tale algoritmo è sufficiente uno stencil compatto e i risultati ottenuti sono paragonabili ai metodi di forzamento diretto classici in termini di accuratezza e robustezza.

3.3.2 Implementazione

L'imposizione delle condizioni al contorno sulla superficie nel metodo dei contorni immersi avviene modificando opportunamente la velocità nei punti della griglia tra il sotto-passo 2 (aggiornamento della velocità) e il sotto-passo 3 (penalità) del solutore. In particolare detta Euleriana la griglia dei punti utilizzata dal solutore, la geometria immersa

è suddivisa in N_t elementi triangolari i cui centroidi, come mostrato in 3.5, sono i punti di una griglia denominata Lagrangiana.

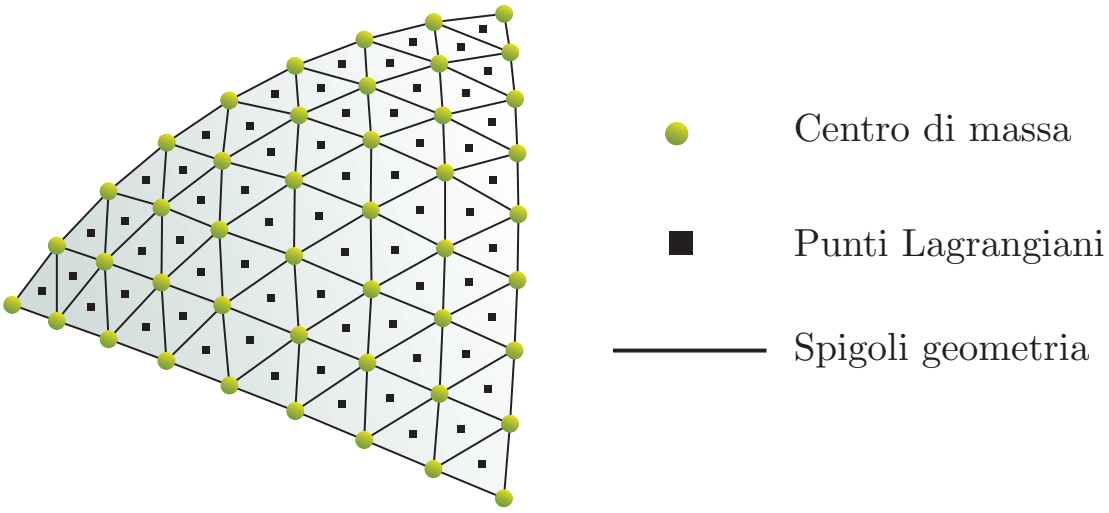


Figura 3.5: *Descrizione della geometria attraverso elementi triangolari i cui centroidi rappresentano la griglia Lagrangiana*

Affinché le condizioni al contorno siano imposte correttamente è necessaria una risoluzione sufficiente della superficie immersa, per questo motivo la distanza tra i punti Lagrangiani deve essere comparabile con quella dei punti Euleriani della griglia di calcolo. In particolare considerando la possibile deformazione dei triangoli in seguito al moto della superficie si ritiene ottimale che la distanza tra i punti Lagrangiani sia circa 0.7 volte quella dei punti Euleriani locali. Si noti che qualora questa condizione non fosse rispettata le superfici immerse risulterebbero porose. Dato un punto Lagrangiano l'algoritmo identifica il punto della griglia Euleriana più vicino, centro della cella di dimensione Δx_i nella i-esima direzione come mostrato in figura 3.6, e crea un dominio di supporto per il punto considerato di estensione $\pm 1.5\Delta x$.
In questo modo N_e (27 in tre dimensioni) punti Euleriani vengono racchiusi nel dominio di supporto e associati al punto Lagrangiano selezionato al quale è anche associato un volume $\Delta V^l = A_l h_l$ dove A_l è l'area dell'l-esimo triangolo e h_l è lo spessore locale, ovvero la dimensione media della mesh nell'intorno del punto selezionato. A questo punto per costruire le funzioni di trasferimento tra le due griglie viene utilizzata l'approssimazione ai minimi quadrati nel modo seguente:

1. La velocità calcolata dal solutore in ogni punto della griglia Euleriana $\hat{\boldsymbol{u}}$ viene passata come input alla routine responsabile dell'imposizione delle condizioni al contorno sulla geometria.

2. La componente $\hat{U}_i$ della velocità in ogni punto l della griglia Lagrangiana è calcolata utilizzando l'approccio ai minimi quadrati mobili (MLS) come:

$$\hat{U}_i(\boldsymbol{x}) = \boldsymbol{p}^T(\boldsymbol{x})\,\boldsymbol{a}(\boldsymbol{x}) = \sum_{j=1}^{m} p_j(\boldsymbol{x})\,a_j(\boldsymbol{x})\,. \tag{3.56}$$

   Dove $\boldsymbol{p}(\boldsymbol{x})$ è il vettore delle funzioni di base con dimensione m, $\boldsymbol{a}(\boldsymbol{x})$ è un vettore di coefficienti a $\boldsymbol{x}$ è la posizione del punto Lagrangiano considerato. Nel codice

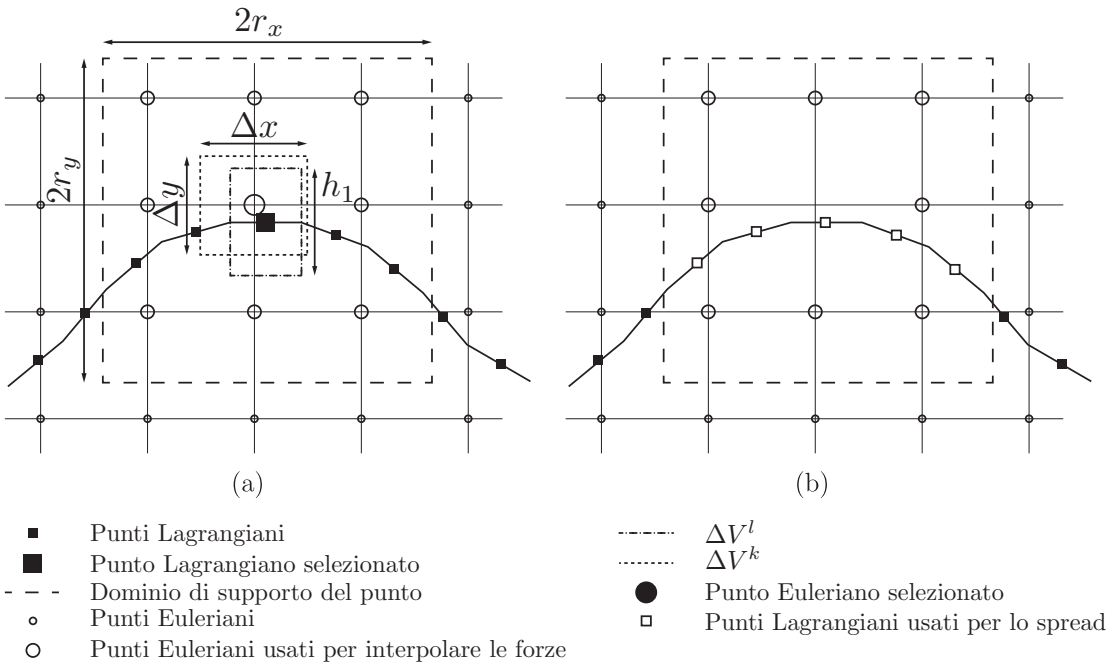


Figura 3.6: *(a) Definizione del dominio di supporto per il punto Lagrangiano considerato, I punti Euleriani appartenenti al dominio di supporto sono coinvolti nell'interpolazione delle variabili nel punto Lagrangiano. (b) Le forze agenti sui punti Lagrangiani interni al dominio di supporto sono diffuse sui punti Euleriani associati al punto considerato*

utilizzato si è scelto di utilizzare funzioni di base lineari: $\boldsymbol{p}^T(\boldsymbol{x}) = \begin{bmatrix} 1, & x, & y, & z \end{bmatrix}$ con $m = 4$ per la capacità, con basso costo computazionale, di rappresentare la variazione nei campi delle variabili di interesse con un'accuratezza paragonabile a quella dello schema di discretizzazione spaziale utilizzato**3VB2009**. Il vettore dei coefficienti $\boldsymbol{a}(\boldsymbol{x})$ è ottenuto minimizzando la norma L^2 pesata definita come:

$$J = \sum_{k=1}^{N_e} W\left(\boldsymbol{x} - \boldsymbol{x}^k\right) \left[\boldsymbol{p}^T\left(\boldsymbol{x}^k\right) \boldsymbol{a}(\boldsymbol{x}) - \hat{u}_i^k\right]^2. \tag{3.57}$$

Dove N_e è il numero di punti della griglia Euleriana che costituiscono il dominio di supporto per determinare i valori nei punti della griglia Lagrangiana, $\hat{u}_i^k$ e $\boldsymbol{x}^k$ sono rispettivamente la componente i-esima di velocità e il vettore posizione nel punto k della griglia Euleriana mentre $W\left(\boldsymbol{x} - \boldsymbol{x}^k\right)$ è una funzione peso assegnata. Tale operazione porta alla definizione dei coefficienti $\boldsymbol{a}$ mediante il sistema lineare

$$\boldsymbol{A}(\boldsymbol{x})\,\boldsymbol{a}(\boldsymbol{x}) = \boldsymbol{B}(\boldsymbol{x})\,\hat{\boldsymbol{u}}_i, \tag{3.58}$$

con

$$\begin{aligned} \boldsymbol{A}(\boldsymbol{x}) &= \sum_{k=1}^{N_e} W\left(\boldsymbol{x} - \boldsymbol{x}^k\right) \boldsymbol{p}\left(\boldsymbol{x}^k\right) \boldsymbol{p}^T\left(\boldsymbol{x}^k\right), \\ \boldsymbol{B}(\boldsymbol{x}) &= \left[W(\boldsymbol{x} - \boldsymbol{x}^1)\,\boldsymbol{p}(\boldsymbol{x}^1) \quad \ldots \quad W\left(\boldsymbol{x} - \boldsymbol{x}^{N_e}\right) \boldsymbol{p}\left(\boldsymbol{x}^{N_e}\right)\right], \\ \hat{\boldsymbol{u}}_i &= \left[\hat{u}_i^1 \quad \ldots \quad \hat{u}_i^{N_e}\right]. \end{aligned} \tag{3.59}$$

Combinando le equazioni 3.56, 3.57 e 3.58 si ottiene:

$$\hat{U}_i(\boldsymbol{x}) = \boldsymbol{\phi}^T(\boldsymbol{x})\,\hat{\boldsymbol{u}}_i = \sum_{k=1}^{N_e} \phi_k^l(\boldsymbol{x})\,\hat{u}_i^k. \tag{3.60}$$

Dove $\boldsymbol{\phi}^T(\boldsymbol{x}) = \boldsymbol{p}^T(\boldsymbol{x})\,\boldsymbol{A}^{-1}(\boldsymbol{x})\,\boldsymbol{B}(\boldsymbol{x})$ è l'operatore di trasformazione da griglia Euleriana a Lagrangiana contenete le funzioni di forma per i punti Lagrangiani l. Per questo lavoro si è utilizzata la stessa funzione peso suggeriti da **3dTP2016**:

$$W\left(\boldsymbol{x}-\boldsymbol{x}^k\right) = \begin{cases} e^{-(r_k/\varepsilon)^2} & r_k \leq 1 \\ 0 & r_k > 1, \end{cases} \tag{3.61}$$

con $\varepsilon = 0.3$ e

$$r_k = \frac{\left|\boldsymbol{x}-\boldsymbol{x}^k\right|}{r_i}. \tag{3.62}$$

Dove r_i è la dimensione del dominio di supporto nella direzione i-esima.

3. Viene calcolata la forza di volume in ogni punto della griglia Lagrangiana come:

$$F_i = \frac{V_i^b - \hat{U}_i}{\Delta t} \tag{3.63}$$

dove V_i^b è la componente i-esima della velocità da imporre come condizione al contorno sui punti della griglia Lagrangiana.

4. Le forze calcolate sulla griglia Lagrangiana vengono trasferite sulle $N-e$ celle Euleriane del dominio di supporto utilizzando le stesse funzioni di forma usate per l'interpolazione e imponendo che la forza totale agente sul fluido non cambi passando da una griglia all'altra. La componente i-esima della forza di volume nel punto Euleriano k risulta in questo modo:

$$f_i^k = \sum_{l=1}^{N_l} c_l \phi_k^l F_i^l, \tag{3.64}$$

dove N_l indica il numero di punti Lagrangiani associati al punto Euleriano k, ovvero quelli facenti parte del dominio di supporto del punto Euleriano k. il fattore di scala c_l è calcolato imponendo che la forza imposta al fluido non cambi nel trasferimento:

$$\sum_{k=1}^{N_{e,tot}} f_i^k \Delta V^k = \sum_{l=1}^{N_l} F_i^l \Delta V^l. \tag{3.65}$$

Dove $N_{e,tot}$ è il numero totale dei punti Euleriani forzati, ΔV^k è il volume di ciascuna cella Euleriana su cui è imposta la forzante e ΔV^l è il volume associato alle celle Lagrangiane. Sostituendo la 3.64 nella 3.65 e riarrangiando i termini si ottiene:

$$c_l = \frac{\Delta V^l}{\sum_{k=1}^{N_e} \phi_k^l \Delta V^k}. \tag{3.66}$$

5. La velocità nei punti della griglia Euleriana modificata in modo da tener conto delle condizioni al contorno del corpo immerso nel dominio è infine calcolata come:

$$\boldsymbol{u}^* = \hat{\boldsymbol{u}} + \Delta t \boldsymbol{f}, \tag{3.67}$$

   dove $\boldsymbol{f}$ è la forza di volume nelle celle della griglia Euleriana.

Sebbene l'indagine di questo lavoro sia svolta su una geometria rigida l'implementazione del metodo dei contorni immersi utilizzata consente di simulare corpi elastici modellabili come un sistema di masse e molle. Per una descrizione più accurata di questi aspetti si veda **3dTP2016**.

CAPITOLO 4

OpenFOAM: cenni sul funzionamento

Attualemnte la soluzione delle equazioni fluidodinamiche all'interno del gruppo di ricerca è affidata al sempre più diffuso toolbox OpenFOAM (Open Field Operation And Manipulation). Tale strumento mette a disposizione un grande numero di solutori e strumenti per la cfd che lo rendono ad oggi una delle soluzioni di riferimento almeno per l'ambito incomprimibile. In questo capitolo verranno presentati, prima da un punto di vista teorico e poi con alcuni aspetti implementativi, gli algoritmi implementati nei solutori incomprimibili standard di OpenFOAM. La parte finale del capitolo è invece dedicata agli aspetti di parallelizzazione e alla relativa efficienza.

4.1 Algoritmi utilizzati dai solutori di OpenFOAM

4.1.1 Notazione

Gli algoritmi su cui si basano i solutori incomprimibili di OpenFOAM sono metodi a correzione di pressione concettualmente compresi nella classe presentata nella sezione 3.1. Tuttavia nel contesto della discretizzazione a volumi finiti che è quella utilizzata da OpenFOAM, è conveniente scrivere l'equazione della quantità di moto come:

$$\boldsymbol{A}\boldsymbol{u} = \sum_{nb} \boldsymbol{A}_{nb}\boldsymbol{u}_{nb} + \boldsymbol{b} + \widetilde{\boldsymbol{\nabla}p}. \tag{4.1}$$

Questa notazione **4FPS2020** è molto potente consentendo di scrivere le equazioni senza specificare gli schemi di approssimazione temporali e spaziali che possono essere diversi per ogni termine dell'equazione. In particolare la 4.1 esprime la velocità nel volume di controllo considerato $\boldsymbol{u}$ come funzione delle velocità delle celle appartenenti all'intorno $\boldsymbol{u}_{nb}$, del contributo della pressione $\widetilde{\boldsymbol{\nabla}p}$ e di un generico termine sorgente $\boldsymbol{b}$ che tiene conto di tutti i termini espliciti dell'equazione (per esempio termini relativi ai passi

temporali precedenti o alle grandezze turbolente). La tilde nel termine di pressione indica una generica discretizzazione dell'operatore gradiente. Allo stesso modo l'equazione di continuità può essere scritta:

$$\sum_n S_n \boldsymbol{u}_n \cdot \boldsymbol{n}_n = 0. \tag{4.2}$$

Dove l'indice della sommatoria n rappresenta le varie facce di area S_n e normale $\boldsymbol{n}_n$ del volume di controllo considerato. Con questa notazione l'equazione della pressione ricavata analogamente a quanto fatto nel capitolo 3 risulta della forma:

$$a\, p = \sum_{nb} a_{nb} p_{nb} + b. \tag{4.3}$$

Dove p è la pressione e con nb si indica l'intorno della cella considerata.

4.1.2 L'algoritmo SIMPLE

L'algoritmo SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) descritto per la prima volta in **4PS1972**, è un metodo a correzione di pressione di tipo standard la cui unica particolarità è il modo in cui viene corretto il campo di velocità. Definendo i campi di pressione e velocità come:

$$p = p^* + p', \tag{4.4}$$

$$\boldsymbol{u} = \boldsymbol{u}^* + \boldsymbol{u}'. \tag{4.5}$$

Dove p^* rappresenta il campo di pressione iniziale, ovvero quello del passo temprale precedente che deve essere corretto con il campo incognito p' per ottenere il campo di pressione p che soddisfi le equazioni incomprimibili di Navier-Stokes. Analogamente $\boldsymbol{u}^*$ rappresenta il campo di velocità predetto, ovvero quello calcolato risolvendo la sola equazione della quantità di moto con il termine di pressione non aggiornato che deve essere corretto con il campo $\boldsymbol{u}^*$ per ottenere il campo di velocità a divergenza nulla $\boldsymbol{u}$. A questo punto se sottraiamo dalla 4.1 la stessa equazione scritta per i campi non corretti (*) otteniamo:

$$\boldsymbol{A}\boldsymbol{u}' = \sum_{nb} \boldsymbol{A}_{nb} \boldsymbol{u}'_{nb} + \widetilde{\boldsymbol{\nabla}} p'. \tag{4.6}$$

Notiamo che per calcolare il termine $\sum_{nb} \boldsymbol{A}_{nb}\boldsymbol{u}'_{bn}$ è necessaria la velocità $\boldsymbol{u}'$ nelle celle adiacenti che, a sua volta per essere calcolata, necessiterebbe quella delle celle adiacenti e così via. In poche parole per correggere la velocità in un punto sarebbe necessario considerare l'intero campo di moto che risulterebbe più costoso del risolvere direttamente il problema accoppiato rendendo il metodo numerico praticamente inutile. Per questo motivo nell'algoritmo SIMPLE tale termine viene eliminato dall'equazione e la correzione della velocità viene effettuata con l'equazione:

$$\boldsymbol{A}\boldsymbol{u}' = \widetilde{\boldsymbol{\nabla}} p', \tag{4.7}$$

che può essere riscritta, per evidenziare la correzione come:

$$\boldsymbol{u} = \boldsymbol{u}^* + \boldsymbol{d}\widetilde{\boldsymbol{\nabla}} p'. \tag{4.8}$$

Dove $d_i = \frac{1}{A_i}$. L'eliminazione del termine indicato può sembrare avere conseguenze disastrose sulla soluzione ottenuta con il metodo, tuttavia non è così, infatti è necessario tenere presente che la soluzione ottenuta dopo la rimozione del termine, rispetta comunque la forma discretizzata delle equazioni di massa e quantità di moto poiché la correzione è uno step intermedio e non interferisce con la struttura delle equazioni. Il dettaglio della costruzione della correzione quindi non interviene sulla convergenza **4DR1984** o meno del metodo quanto tuttavia sul rate di quest'ultima. Una dettagliata trattazione del problema è data in **4P1980**. L'eliminazione del termine considerato è anche l'origine della prima parte dell'acronimo del metodo: "semi-implicit". Infatti, sebbene la discretizzazione utilizzata dal metodo possa essere completamente implicita, lo schema risultante, poiché omette l'influenza dell'intero campo di moto sulla correzione della velocità locale è detto semi implicito.
Una volta definita la correzione della velocità l'algoritmo segue il procedimento logico dei metodi a correzione di pressione e, in particolare, si articola nelle seguenti fasi:

1. Dato il campo di pressione al tempo/iterazione precedente(o condizione iniziale) p^*.

2. Viene aggiornato il campo di velocità per ottenere $\boldsymbol{u}^*$ risolvendo l'equazione della quantità di moto del tipo 4.1 utilizzando come campo di pressione p^* :

$$\boldsymbol{A}\boldsymbol{u}^* = \sum_{nb} \boldsymbol{A}_{nb}\boldsymbol{u}^*_{nb} + \boldsymbol{b} + \widetilde{\boldsymbol{\nabla}p^*}. \tag{4.9}$$

3. Viene risolta l'equazione della pressione per ottenere il campo p':

$$a\,p' = \sum_{nb} a_{nb}p'_{nb} + b. \tag{4.10}$$

4. Il campo di pressione corretto è calcolato aggiungendo p' al campo iniziale p^*:

$$p = p^* + p'. \tag{4.11}$$

5. La velocità viene corretta utilizzando la relazione:

$$\boldsymbol{u} = \boldsymbol{u}^* + \boldsymbol{d}\widetilde{\boldsymbol{\nabla}p'}. \tag{4.12}$$

6. Vengono risolte altre eventuali equazioni di conservazione (per esempio quelle relative alle quantità turbolente).

7. il campo di pressione p^* viene sostituito da p e l'algoritmo viene iterato fino a convergenza.

4.1.3 L'algoritmo PISO

L'algoritmo PISO (Pressure-Implicit with Splitting of Operators) presentato per la prima volta in **4IR1986**, nella sua versione incomprimibile risulta molto simile all'algoritmo SIMPLE nonostante sia concettualmente diverso. Tale algoritmo si basa infatti sull'operator-splitting applicato alle variabili pressione e velocità il cui accoppiamento è garantito da una serie di step di tipo previsione-correzione. In particolare definiti i campi intermedi identificati con gli apici: *, ** e *** l'algoritmo si articola nelle seguenti fasi:

1. *predizione:* il campo di pressione p^n del passo temprale precedente è utilizzato come guess iniziale e utilizzando tale valore viene risolta l'equazione della quantità di moto del tipo 4.9
$$\boldsymbol{A}\boldsymbol{u}^* = \sum_{nb} \boldsymbol{A}_{nb}\boldsymbol{u}^*_{nb} + \widetilde{\boldsymbol{\nabla} p^n}. \tag{4.13}$$
Dove si noti che il campo ottenuto $\boldsymbol{u}^*$ non soddisfa la condizione di divergenza nulla.

2. *prima correzione:* A questo punto si introduce un campo di pressione p^* e il corrispondente campo di velocità aggiornato che soddisfi la condizione di divergenza nulla $\boldsymbol{u}^{**} = 0$. Per tale campo viene scritta l'equazione della quantità di moto
$$\boldsymbol{A}\boldsymbol{u}^{**} = \sum_{nb} \boldsymbol{A}_{nb}\boldsymbol{u}^*_{nb} + \boldsymbol{b} + \widetilde{\boldsymbol{\nabla} p^*}. \tag{4.14}$$
Dove di fondamentale importanza è che i termini incogniti siano solo quello a sinistra e il campo di pressione, mentre gli altri termini contenti la velocità siano descritti con il campo noto $\boldsymbol{u}^*$. Tale assunzione risulta praticamente equivalente alla scelta di trascurare il termine $\sum_{nb} \boldsymbol{a}_{nb}\boldsymbol{u}'_{bn}$ nell'equazione di correzione della velocità 4.6. Infatti applicando l'operatore di divergenza alla 4.14 il termine a sinistra dell'uguale sparisce per definizione e resta un'equazione per la pressione formalmente analoga alla 4.10:
$$a\, p^* = \sum_{n} a_n p^*_n + b. \tag{4.15}$$
Dove, essendo noto $\boldsymbol{u}^*$ i termini relativi sono stati raccolti in b. Il risultato è un equazione implicita nella sola incognita p^* che, una volta risolta consente di determinare anche il campo $\boldsymbol{u}^{**}$ grazie alla 4.14.

3. *seconda correzione:* Le operazioni fatte nella prima correzione possono essere ripetute introducendo un campo di pressione p^{**} e il corrispondente campo a divergenza nulla u^{***}. Che possono essere determinati analogamente a quanto fatto in precedenza risolvendo prima l'equazione per la pressione
$$a\, p^{**} = \sum_{n} a_n p^{**}_n + b, \tag{4.16}$$
e poi quella della quantità di moto:
$$\boldsymbol{A}\boldsymbol{u}^{***} = \sum_{nb} \boldsymbol{A}_{nb}\boldsymbol{u}^{**}_{nb} + \boldsymbol{b} + + \widetilde{\boldsymbol{\nabla} p^{**}}. \tag{4.17}$$

4. *avanzamento temporale* Il solutore avanza al passo temprale successivo e utilizza come guess iniziale i campi determinati.

La formulazione con doppia correzione è quella presentata dagli autori nell'articolo originale e il metodo in questo modo risulta accurato al secondo ordine, con delle limitazioni sul passo temprale dovute al fatto che l'equazione del momento negli step di correzione risulta esplicita.

4.2 I solutori incomprimibili standard di OpenFOAM

In questa sezione vengono presentati i solutori incomprimibili presenti nella versione OpenFOAM 5.0 che è stata utilizzata per questo lavoro. Si tenga presente che in versioni successive i solutori incomprimibili sono stati unificati nascondendo la scelta tra l'uno e l'altro nei parametri dei dizionari.
I solutori incomprimibili standard implementati su OpenFOAM sono basati sui due algoritmi presentati e ne seguono in modo abbastanza fedele gli step tuttavia ci sono due aspetti che è utile citare: l'interpolazione di Rhie-Chow e la correzione di non-ortogonalità.
Per questioni di flessibilità, generalità ed efficienza, OpenFOAM utilizza una griglia co-locata, ovvero tutte le grandezze fluidodinamiche sono definite in un solo punto all'interno di ciascun volume di controllo: il centroide. Ciò implica che per evitare le oscillazioni ad alte frequenze del campo di pressione con conseguente instabilità a scacchiera **4P1980** si debba ricorrere a metodi di correzione ad hoc. Il più famoso che è anche quello implementato in OpenFOAM è il metodo di Rhie-Chow descritto in **4ZZB2014**. L'implementazione di tale metodo in OpenFOAM, come descritto dettagliatamente in **4K2006**, è in realtà di tipo implicito nel senso che avviene sostanzialmente grazie all'utilizzo del teorema della divergenza per il calcolo dei termini legati alla pressione. In questo modo infatti, il termine relativo al gradiente di pressione è calcolato come sommatoria delle pressioni sulle facce di ciascuna cella mentre il termine laplaciano nell'equazione della pressione usa il valore del gradiente di pressione calcolato sulle facce della cella considerata. Ciò ha come conseguenza che il gradiente in questo caso è calcolato utilizzando i centroidi delle due celle che la faccia considerata divide. In questo modo è come se le variabili venissero prese in punti diversi della griglia e quello che si ottiene è un campo di pressione e di velocità senza oscillazioni in linea con la correzione di Rhie-Chow anche se quest'ultima non è presente esplicitamente.
Il secondo aspetto riguarda la soluzione dell'equazione della pressione negli algoritmi visti, in particolare tale equazione viene risolta più volte di fila (in base al coefficiente "nNonOrtogonalCorr" specificato) ogni volta utilizzando il valore di pressione ottenuto dalla soluzione precedente. Tale passaggio è inutile per griglie ortogonali ma diventa importante per mesh di problemi tipici dell'ingegneria che risultano spesso localmente non perfettamente ortogonali.

4.2.1 Il solutore SimpleFOAM

SimpleFOAM **4OFWsimple**, **4J1996**, **4MMD2015** è un solutore per correnti incomprimibili, stazionarie, turbolente o laminari che implementa l'algoritmo SIMPLE in modo sostanzialmente identico a quello presentato ma con una differenza degna di nota: il fattore di rilassamento. Come accennato in precedenza il trascurare il termine $\sum_{nb} \boldsymbol{A}_{nb}\boldsymbol{u}'_{bn}$ non compromette il funzionamento generale dell'algoritmo ma ne modifica le proprietà di convergenza, in particolare è possibile che il metodo abbia una convergenza lenta se non addirittura che diverga per alcuni problemi di interesse pratico. Per garantire una stabilità superiore al metodo, l'autore stesso **4PS1972** consiglia quindi di inserire un fattore di rilassamento α nella correzione della pressione modificando la 4.11 in:

$$p = p^* + \alpha p', \tag{4.18}$$

con valori suggeriti: $0.5 < \alpha < 0.8$ dove si tenga presente che più basso è il coefficiente di rilassamento più il metodo è stabile ma più bassa è la velocità di convergenza ovvero sono necessarie più iterazioni del metodo per arrivare alla soluzione. Nell'implementazione in SimpleFOAM anche l'equazione della quantità di moto è rilassata e invece della 4.9 è risolta:

$$\frac{1}{\alpha}\boldsymbol{A}\boldsymbol{u}^* = \sum_{nb} \boldsymbol{A}_{nb}\boldsymbol{u}^*_{nb} + \boldsymbol{b} + \boldsymbol{S}\Delta\boldsymbol{p}^* + \frac{1-\alpha}{\alpha}\boldsymbol{A}\boldsymbol{u}^{n-1}. \tag{4.19}$$

Dove $\boldsymbol{u}^{n-1}$ è il campo di velocità dell'iterazione precedente. Si noti che l'introduzione del fattore di rilassamento fa perdere significato al tempo della simulazione, il che è accettabile per problemi stazionari o di transitorio fittizio ma può essere fonte di errori grossolani per studi su fenomeni di transitori reali

4.2.2 Il solutore PisoFOAM

PisoFOAM **4OFWpiso** è un solutore per correnti incomprimibili, instazionarie, turbolente o laminari che implementa l'algoritmo PISO in modo equivalente a quello presentato nella descrizione dell'omonimo algoritmo. Anche in questo caso è necessaria l'implementazione dell'interpolazione di Rhie-Chow lavorando OpenFOAM su griglie coo-locate. Il numero di correzioni dell'equazione della quantità di moto di default è due come nell'algoritmo proposto originariamente, tale numero è però modificabile a seconda delle necessità dell'utente.

4.2.3 Il solutore IcoFOAM

IcoFOAM **4OFWico** è un solutore per correnti incomprimibili, instazionarie, laminari che implementa l'algoritmo PISO nello stesso modo del solutore PisoFOAM, tuttavia al contrario di questo non presenta i termini relativi alla turbolenza nell'equazione del momento ed è quindi utilizzabile solo per correnti laminari.

4.2.4 Il solutore PimpleFOAM

PimpleFOAM **4OFWpimple** è un solutore per correnti incomprimibili, instazionarie, laminari o turbolente che unisce le idee dell'algoritmo PISO e SIMPLE. In particolare tale solutore implementa il metodo PISO con il numero di correzioni impostabile dall'utente mediante il parametro "nCorrectors" come PisoFOAM e IcoFOAM ma in più permette di ripetere, all'interno dello stesso passo temprale, il ciclo PISO più volte (controllabile dal parametro "nOuterCorrectors"). Naturalmente impostando quest'ultimo parametro a uno (che è, tra l'altro, il valore di default) questo solutore è del tutto identico a PisoFOAM tuttavia aumentandone il valore ogni timstep è formato da "nOuterCorrectors" iterazioni dell'algoritmo PISO. É necessario tenere presente che, per lo studio di un transitorio reale dove si voglia quindi mantenere il tempo non dilatato non è possibile rilassare le equazioni finali di ogni passo temprale ma quelle all'interno di ogni sotto-ciclo si. Per questo motivo il solutore PimpleFOAM risulta efficiente ed adatto per i problemi mal inizializzati, di difficile convergenza o per i quali si vuole utilizzare un alto numero di Courant **4H2019**.

4.3 Parallelizzazione in OpenFOAM

La parallelizzazione di OpenFOAM per i problemi fluidodinamici è bastata sulla decomposizione del dominio gestita dall'utility "decomposePar" ed è implementata attraverso la libreria MPI (Message Passing Interface) come nel codice presentato nel capitolo 3. La natura flessibile e generale della libreria OpenFOAM rende difficile lo studio della scalabilità degli algoritmi, infatti l'implementazione delle varie componenti e modulare tanto che i solutori di OpenFOAM sono scritti come fossero codici seriali poiché la parallelizzazione è implementata ad un livello più alto che consente di essere applicata a ogni solutore scelto. Per questo motivo, sopratutto quando si parla di scalabilità massiva ovvero su sistemi con più di 1000 core, è difficile determinare una stima teorica su quale sia l'effettiva scalabilità del codice utilizzato. In letteratura sono presenti diversi lavori **4AR2016**, **4SR2015**, **4DCP2015**, **4S2014** che testimoniano, su casi diversi e con solutori diversi, che su architetture ottimizzate e realizzate per il calcolo parallelo sia possibile raggiungere uno scaling lineare fino ad almeno 1000 core. Un esempio del risultato ottenuto da **4DCP2015** è mostrato in figura 4.1

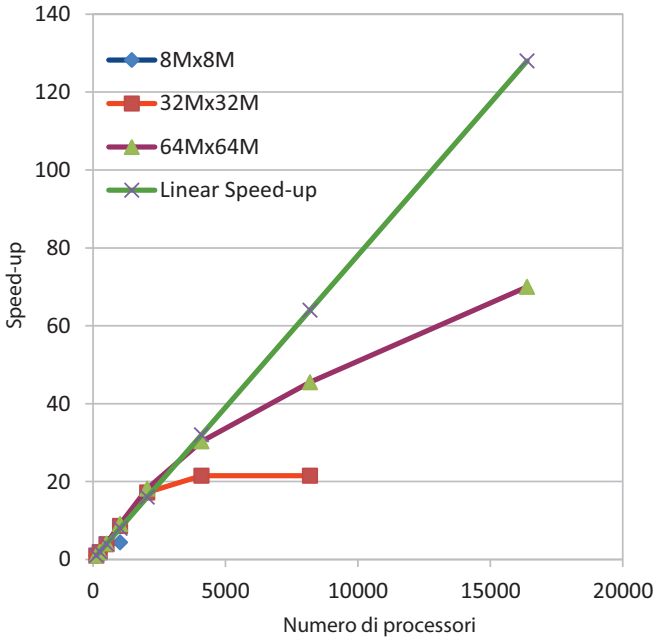


Figura 4.1: *Speedup parallelo al crescere della dimensione del problema ottenuto su una simulazione DNS con il solutore IcoFOAM* **4DCP2015**. *I valori riportati in legenda indicano la dimensione delle matrici quadrate che descrivono i sistemi lineari dei tre casi testati, per esempio il primo caso è ricondotto a un sistema lineare di dimensione 8 milioni.*

Da notare che tale risultato è un limite teorico dell'algoritmo ma è dipendente dall'hardware su cui esso è eseguito nonché dall'implementazione di MPI utilizzata che, per raggiungere i risultati migliori con un grande numero di processori deve essere ottimizzata per l'architettura utilizzata.
Un aspetto critico è la dipendenza dell'efficienza di parallelizzazione per OpenFOAM e per tutti i codici basati su griglie conformi è data dalla decomposizione del dominio. Per geometrie complesse può infatti diventare problematico limitare le comunicazioni

necessarie tra i processori nonostante sia l'obiettivo degli algoritmi di decomposizione del dominio. Tale problema è totalmente superato nel codice DNS utilizzato che lavora su una griglia parallelepipeda di facile decomposizione.

CAPITOLO 5

Set-Up simulazioni

In questo capitolo viene descritto il set-up utilizzato per confrontare le prestazione dei due codici descrivendo le scelte effettuate per ottenere un caso realizzabile con entrambe le strategie di soluzione. Vengono presentati inoltre gli step fondamentali per arrivare a lanciare la simulazione con ciascuno dei due codici dando una descrizione pratica sulla scelta e modifica dei parametri principali.

5.1 La scelta del test case

La scelta del caso da analizzare per il confronto è stata fatta tenendo in considerazione da un lato l'esigenza del gruppo di ricerca di indagare l'accelerazione dei tempi di calcolo per casi di interesse e dall'altro i limiti dei codici utilizzati. In particolare poiché OpenFOAM risulta nativamente flessibile ed è già utilizzato all'interno della procedura di risoluzione i limiti del set-up sono stati dettati dal solutore di Guermond. La geometria utilizzata per la simulazione, mostrata in figura 5.1 è ottenuta a partire da una TAC di un paziente e quindi mantiene la complessità geometriche del caso reale tuttavia il dominio di calcolo deve avere forma parallelepipeda poiché come già accennato il solutore di Guermond lavora su griglie di questo tipo.
Questo aspetto è in un certo senso penalizzante per OpenFOAM nel senso che le celle all'esterno del condotto nasale sono per tale codice inutili e comportano un aggravio dei tempi di calcolo che sarebbe facilmente evitabile. Tuttavia ci sono alcuni fattori che hanno portato a questa scelta:

- Il confronto che si vuole tra i due codici è punto-a-punto poiché gli aspetti implementativi del codice di Guermond potrebbero variare nel codice che si valuterà se inserire nella procedura al posto di OpenFOAM, per questo motivo i due solutori devono lavorare sostanzialmente sullo stesso numero di celle e su casi il più simili possibile.

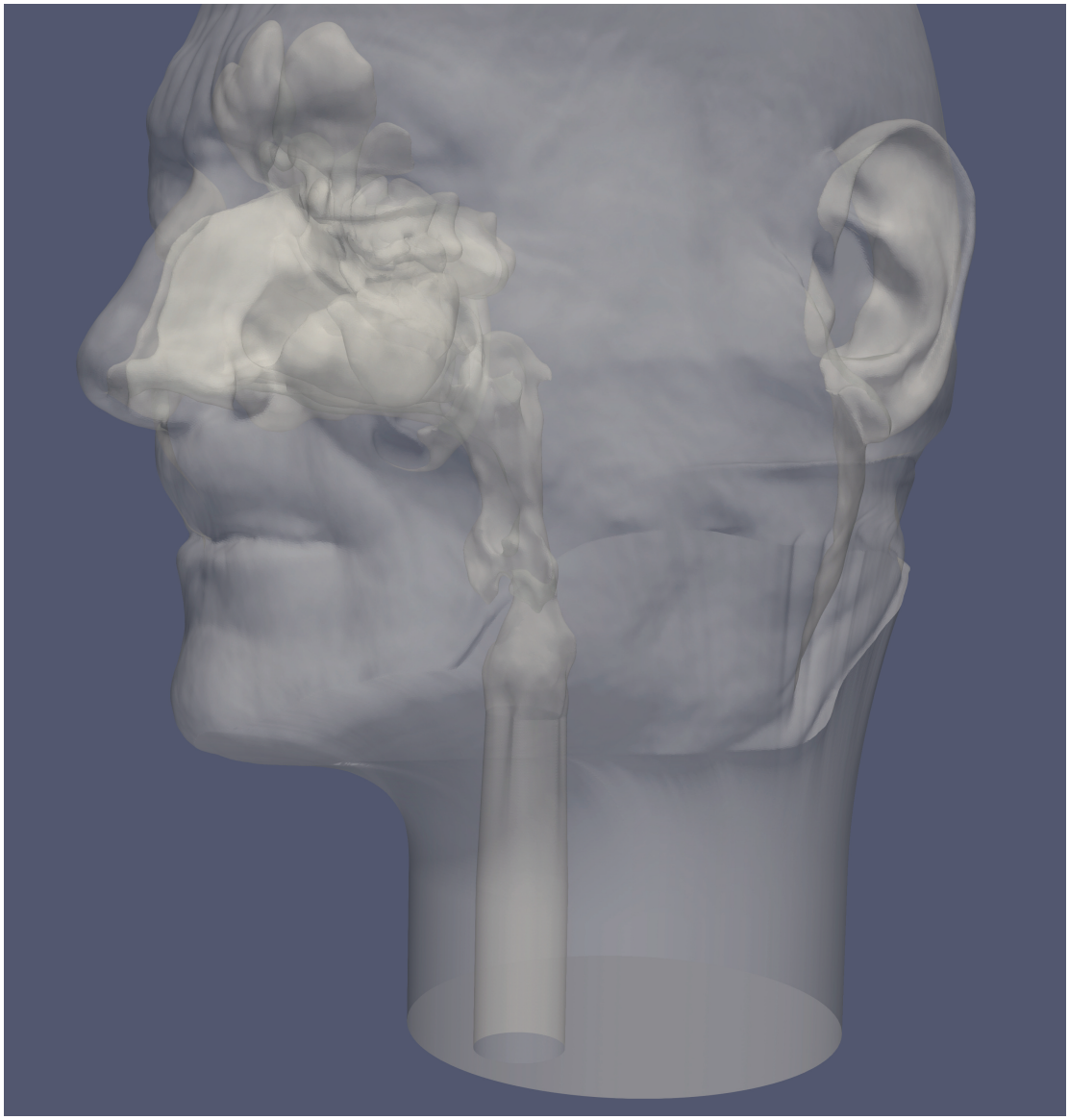


Figura 5.1: *Geometria del caso analizzato*

- la soluzione ottenuta non è rilevante per il lavoro in questione purché la soluzione sia convergente poiché l'obiettivo è valutare l'efficienza dei codici

- La maggior parte dei punti (oltre il'90%) nella mesh di OpenFOAM sono all'interno del condotto, quindi i punti inutilizzati, considerando che un minimo di punti esterni sono necessari per l'imposizione delle condizioni al contorno, sono inferiori al 10%. Sebbene tale aggravio non risulti estremamente significativo andrà comunque considerato nel confronto di prestazioni e confrontato con l'eventuale guadagno.

Per quanto riguarda le condizioni al contorno con riferimento alla figura 5.2 tutte le facce del dominio parallelepipedo sono trattate come pareti solide e lo stesso vale per la geometria inserita nel dominio.

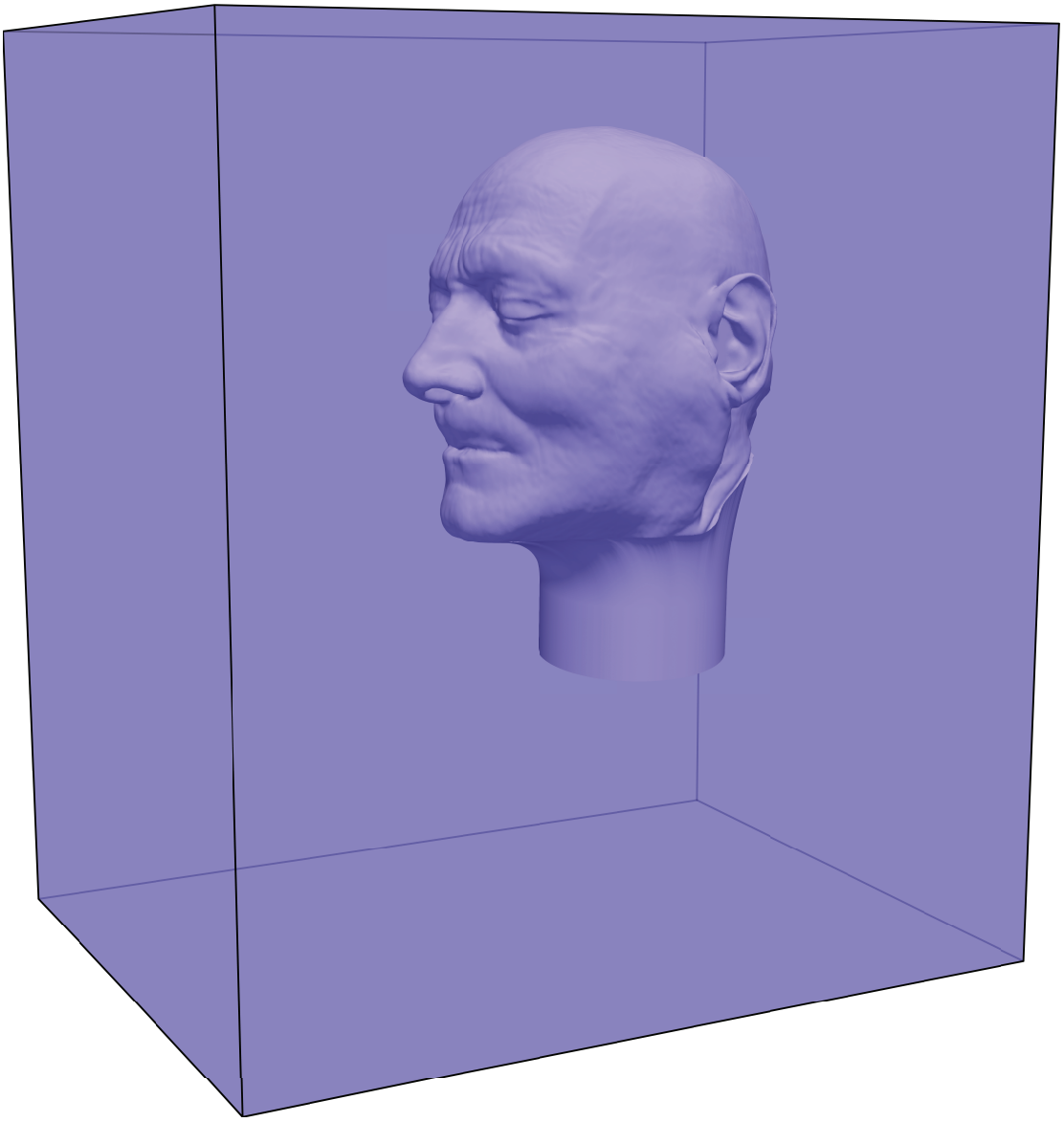


Figura 5.2: *Schematizzazione delle condizioni al contorno del caso analizzato*

Il forzamento del flusso avviene quindi non attraverso le condizioni al contorno sulle facce dalle quali non avviene ne ingresso ne uscita di massa ma mediante l'utilizzo di una forza di volume inserita in un prolungamento fittizio della trachea. Questa scelta che si discosta dai casi risolti fin ora con OpenFOAM deriva dalla particolare implementazione del metodo dei contorni immersi nel codice di Guermond che non consente intersezioni tra la geometria immersa e le pareti del dominio di calcolo. I dettagli dell'imposizione della forza di volume necessaria per mettere in moto il fluido verranno presentati per ciascun software tuttavia il concetto è inserire una forza di volume che crei nella trachea una velocità fisicamente sensata, in particolare si è scelto, in accordo con la letteratura citata nel capitolo 2 e con i risultati già ottenuti il altre simulazioni dal gruppo di ricerca, il valore di $1.1\,m/s$. La scelta della "scatola" come dominio non è nuovo nella storia del gruppo di ricerca, infatti è stato utilizzato dal lavoro sperimentale **5CFQ2017** nel quale

il set-up sperimentale prevedendo l'utilizzo di un metodo ottico imponeva la scelta di materiali con uguale indice di rifrazione. Come fluido al posto dell'aria, per raggiungere l'indice del silicone utilizzato per realizzare il modello, si è usata una miscela di acqua e glicerina che era riempiva un box appositamente realizzato.

5.2 Set-up della simulazione con il codice di Guermond

5.2.1 Generalità sul Software

Il codice utilizzato è scritto nel linguaggio fortran 90 in modo molto modulare, in modo che ogni aspetto del codice sia contenuto in un file diverso e possa essere modificato senza alterare il resto dei componenti. I file sono organizzati in quattro directory:

- CODE: è la directory principale nella quale viene eseguito il codice, contiene oltre all'eseguibile e ai file di mesh e configurazione alcuni sorgenti che è utile poter modificare, in particolare il programma principale *pCNST.f90* e il file che contiene la definizione delle condizioni al contorno e di eventuali forze di volume *solutions.f90*. In questa cartella è anche contenuto il makefile che permette di ricompliare il software, notare che tale azione è necessaria ogni volta che si modifica il codice sorgente (qualsiasi file .f90).
- SOURCES: questa cartella contiene la maggior parte delle routine del codice: implementazione del metodo dei contorni immersi, definizione dei tipi e della struttura dei dati utilizzati, algoritmi di soluzione veri e propri e tutte le principali routine necessarie per l'implementazione dei metodi presentati.
- Tools: contiene routine utili per rifinire la mesh per il metodo dei contorni immersi se necessario
- UTILITIES: contiene routine utili per la scrittura dei risultati ottenuti in formato .vtk

Poiché il codice è composto da un grande numero di file e questo lavoro non ha come scopo la descrizione dettagliata del funzionamento del codice verranno citati e dettagliati solo i file i le parti di codice che è necessario considerare e modificare per eseguire simulazioni standard. Oltre ai sorgenti ci altri file contenuti nella cartella CODE importanti per il set-up della simulazione, tra questi la definizione della geometria da inserire e i file di mesh. Di particolare importanza è il file *data* che contiene i parametri fondamentali per la simulazione, la sua struttura tipica è:

```
1  ===Space dimension
2  3
3  ===Time step
4  0.00001
5  ===Final time
6  0.001
7  ===Fluid Reynolds number
8  66666
9  ===Thermal diffusivity
10 1.
11 ===Xi coefficient (projection)
12 0.5
```

```
===Mesh uniform? true/false
.FALSE.
===size of box: Lx
350
===size of box: Ly
450
===size of box: Lz
500
===Number of procs in x
1
===Number of procs in y
1
===Number of procs in z
1
===Number of interior points per proc in x
218
===Number of interior points per proc in y
280
===Number of interior points per proc in z
312
===Velocity BC in x direction at 0 and Lx (1=Dir,2=Neu)
1 1
===Velocity BC in y direction at 0 and Ly (1=Dir,2=Neu)
1 1
===Velocity BC in z direction at 0 and Lz (1=Dir,2=Neu)
1 1
===Scalar BC in x direction at 0 and Lx (1=Dir,2=Neu)
1 1
===Scalar BC in y direction at 0 and Ly (1=Dir,2=Neu)
1 1
===Scalar BC in z direction at 0 and Lz (1=Dir,2=Neu)
1 1

============================Booleans
===Analytic test? (True/False)
.TRUE.
===Source term in momemtum? (True/False)
.TRUE.
===Nonlinear term term in momemtum? (True/False)
.TRUE.
===Solve for passive scalar? (True/False)
.FALSE.
===Restart from file? (True/False)
.FALSE.
===Store for restart file? (True/False)
.TRUE.

============================Postprocessing
===History (global index of grid point to be sampled if /=0)
5
===Cartesian coordinates of proc where history is stored
1 1 2
===Number of slices
0                  # nb_slices.  If >0 next rows contain the slice data
===Time period for sampling
1.                 # time period of sampling for each slice
===Index of orthogonal direction for each slice
3                  # direction orthogonal to each slice
===Coordinate of the proc in the given direction
```

```
72 0                    # coordinate of the blocks in the given direction
73 ===Coordinate of the slice in the proc
74 21                   # coordinate of the slice in the given block
```

Space dimension = 3 indica che si tratta di un problema a tre dimensioni, la simulazione inizia dal tempo zero e avanza fino al *Final time* con passo indicato dal *Time step*, il Reynolds della simulazione è da intendersi come reciproco della viscosità dell'equazione della quantità di moto mentre il coefficiente Ξ, come discusso nel capitolo 3, ha effetto sulla convergenza del metodo. Il flag booleano *Mesh uniform* indica se i punti della mesh hanno distribuzione uniforme nelle tre dimensioni nel qual caso non è necessario fornire le coordinate dei punti di griglia che verranno calcolati da un'apposita routine. Le dimensioni del dominio, il numero di processori in ogni direzione e il numero di punti per precessore nelle varie direzioni sono indicate dalle omonime flags. Si tenga presente che il numero di processori totali utilizzati sarà $N_p = N_x \cdot N_y \cdot N_z$ mentre il numero di punti interni totali della griglia $N_{pti} = N_p \cdot N_{ptix} \cdot N_{ptiy} \cdot N_{ptiz}$ ai quali vanno aggiunti quelli per tener presente delle interfacce come verrà discusso nel dettaglio più avanti. Il tipo di condizioni al contorno per la velocità ed un eventuale scalare passivo è indicato per le sei facce del dominio utilizzando i flag 1 per le condizioni di Dirichlet e 2 per quelle di Neumann. Gli altri flags sono meno importanti e abbastanza autoesplicativi, da menzionare *Restart from file* che permette di ricominciare una simulazione dai file .rest dei una simulazione precedente. Per fare questo è necessario impostare il flag su .TRUE. e rinominare i file di restart di ogni processore da (per esempio per il processore 0 0 0) *proc_0_0_0.rst* a *proc_0_0_0_odd.rst*.

5.2.2 Generazione dell'stl

La geometria utilizzata è stata generata a partire da quella mostrata in 5.3 già utilizzato per precedenti simulazioni OpenFOAM del gruppo ed ottenuto direttamente della TAC di un paziente attraverso la procedura descritta nel capitolo tre della tesi **5LM2014**.
Alcune modifiche sono risultate necessarie per ottenere una geometria compatibile anche con il codice di Guermond che, in particolare necessita un file in formato .gts (Gnu Triangulates Surface) che descriva una superficie chiusa, costituita da triangoli il più possibile equilateri di lato circa 0.7 la dimensione delle celle della mesh di volume. A partire dalla geometria iniziale si è in primo luogo dovuto creare artificialmente un prolungamento della trachea e del collo per ottenere una zona artificiale dove inserire la forzante in modo da non influire sul flusso nella zona di interesse e contemporaneamente si è ritenuto opportuno eliminare le clavicole sia perché le estremità appuntite causavano instabilità nel codice con la necessità di ridurre la dimensione delle celle, sia perché ritenute inutili ai fini della simulazione. Successivamente è stato necessario ri-triangolare l'intera superficie per ottenere una superficie unica che è stata poi regolarizzata in modo da rispettare i criteri indicati. La geometria così ottenuta è stata posizionata in un sistema di coordinate analogo a quello del codice di Guermond come mostrato in figura 5.4 in modo che le coordinate dei punti fossero direttamente quelle corrette nel dominio. La superficie è stata infine salvata in formato .stl ed analizzata con test di diagnosi di qualità utilizzando MeshLab e poi convertita nel formato .gts utilizzando l'utility stl2gts eseguibile, una volta installata su linux, attraverso il comando: *stl2gts [OPTIONS] < input.stl > output.gts*.
Per fare in modo che il codice veda la geometria è necessario mettere il file .gts nella cartella CODE insieme ad un apposito file *surface.in* con il seguente formato:

Figura 5.3: *Stl di partenza della geometria utilizzata per le simulazioni*

```
1 3
'LugaresiFullNewR'
.FALSE. 0

```

Dove la prima riga contiene il numero di superfici da includere e dimensionalità dello spazio considerato (nel caso in esame una sola superficie tridimensionale), la seconda riga contiene tra apici il nome (senza estensione) del file .gts che contiene la superficie da inserire infine la terza riga contiene due flag: uno booleano per indicare al codice se la superficie si muove (.TRUE.) o è fissa (.FALSE.) e di seguito un intero per sapere se calcolare le forze agenti sulla superficie (0 non vengono calcolate, 1 si). Da notare che se fossero presenti più di una superficie le righe 2 e 3 andrebbero ripetute per ciascuna.

5.2.3 Generazione della mesh

La mesh di volume è generata dal software in maniera automatica nella fase di inizializzazione del solutore a partire dai tre file *xc.dat yc.dat* e *zc.dat* che devono essere collocati nella cartella CODE. Ciascun file ha una struttura del tipo:

```
315
-1.5974440894568689
0.
1.5974440894568689
3.1948881789137378
4.792332268370607
6.3897763578274756
7.987220447284344
9.584664536741213
11.182108626198083
12.779552715654951
14.37699680511182
```

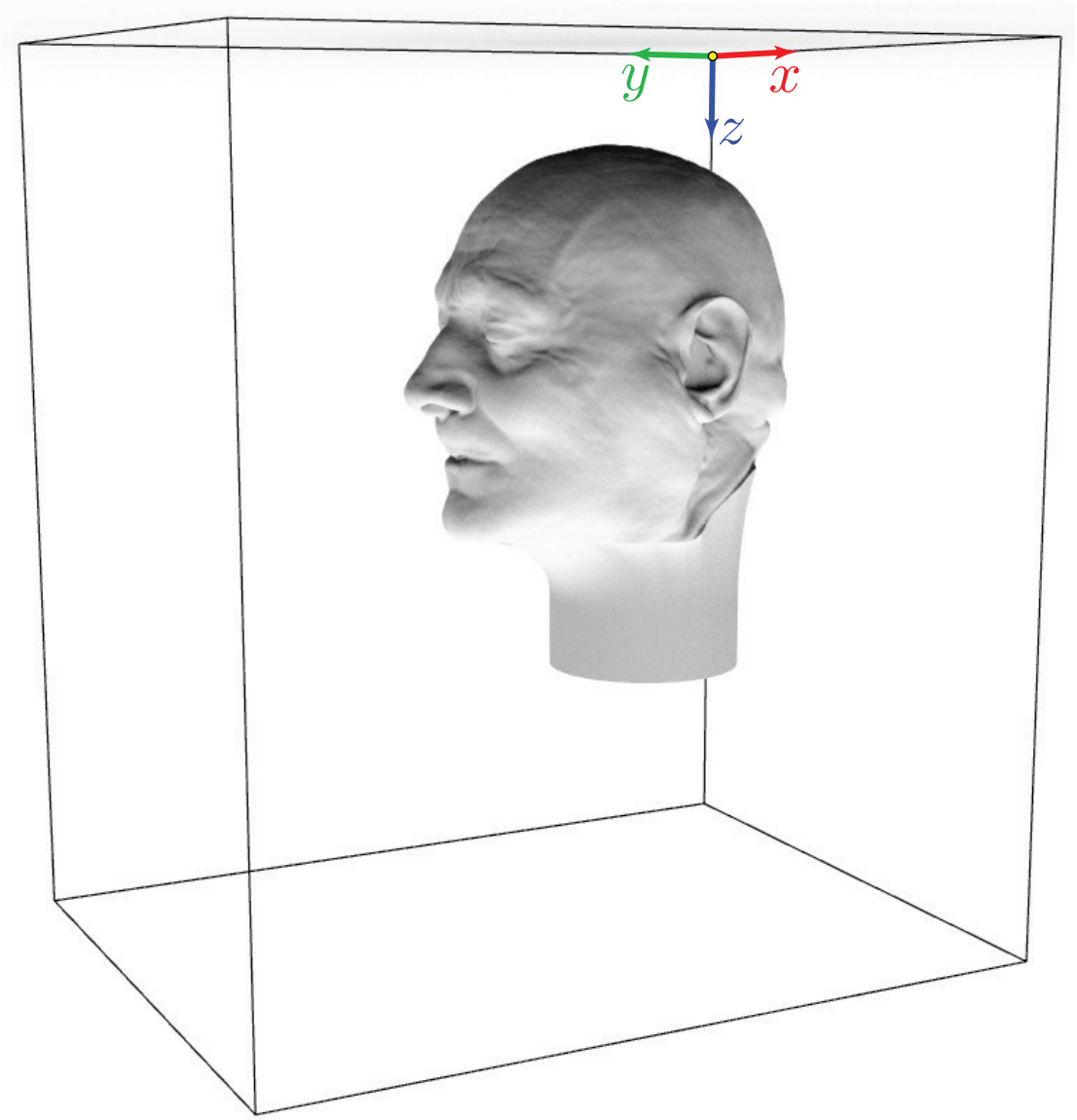


Figura 5.4: *Sistema di coordinate utilizzato dal solutore di Guermond*

```
15.974440894568689

.............

492.01277955271564
493.6102236421725
495.2076677316294
496.80511182108626
498.40255591054313
500.
501.59744408945687
503.19488817891374


```

dove la prima riga contiene il numero di punti interni totali nella direzione x, y o z considerata (a seconda del file) mentre le successive contengono la coordinata omonima al file di ciascun punto della griglia. Da notare che questi file e le indicazioni nel file *data* devono essere compatibili, in particolare considerando per esempio la direzione z, se in questa direzione nel file data è specificato che il numero di punti interni per processore è $n_{ptip_z} = 312$ e il numero di processori è $N_{proc_z} = 1$ la prima riga del file *zc.dat* sarà

$$N_{zc.dat} = (n_{ptip_z} + 1)\, N_{proc_z} + 2 = 315 \tag{5.1}$$

Da notare che i punti definiti in realtà nel file ovvero il numero di righe dopo la prima sono due in più poiché è necessario al software avere punti esterni al dominio di interesse per la gestione delle condizioni al contorno. Nell'esempio analizzato quindi il file conterrà 317 coordinate dopo la prima riga per un totale di 318 righe.
In linea di principio il codice accetta quindi una qualsiasi distribuzione di punti nelle tre direzioni con la possibilità quindi di ottenere raffinamenti localizzati, naturalmente il raffinamento verrà effettuato lungo tutta l'estensione del dominio portando a raffinare zone che magari risulterebbero inutili. Per la creazione dei file di mesh si è utilizzato un script Matematica sia perché ne accelera la creazione sia perché consente di ottenere la distribuzione voluta rapidamente.

5.2.4 Condizioni iniziali e al contorno

Per imporre le condizioni al contorno è necessario, oltre a specificarne il tipo nel file *data* anche modificare il file *solution.f90* definendo opportuni valori per le componenti della velocità o per la sua derivata a secondo del tipo. In particolare si riposta il segmento di codice per l'imposizione delle condizioni del caso test:

```
FUNCTION Dirichlet_u(x,t)
IMPLICIT NONE
REAL(KIND=8),      INTENT(IN)  :: x(3), t
REAL(KIND=8)                   :: Dirichlet_u
REAL(KIND=8)                   :: xf, yf, zf, U0

!= Prescribe boundary conditions
Dirichlet_u = 0.0d0

END FUNCTION Dirichlet_u

!#######################################################!

FUNCTION Dirichlet_v(x,t)
IMPLICIT NONE
REAL(KIND=8),      INTENT(IN)  :: x(3), t
REAL(KIND=8)                   :: Dirichlet_v
REAL(KIND=8)                   :: xf, yf, zf, U0

!= Prescribe boundary conditions
Dirichlet_v = 0.0d0


END FUNCTION Dirichlet_v

!#######################################################!

```

```
28 FUNCTION Dirichlet_w(x,t)
29 IMPLICIT NONE
30 REAL(KIND=8),      INTENT(IN)  :: x(3), t
31 REAL(KIND=8)                   :: Dirichlet_w
32 REAL(KIND=8)                   :: xf, yf, zf, U0
33
34 != Prescribe boundary conditions
35 Dirichlet_w = 0.0d0
36
37 END FUNCTION Dirichlet_w
38
```

Da notare che tali funzioni impongono una condizione globale su tutto il contorno del dominio, per imporre valori diversi su facce diverse è necessario definirli mediante un ciclo if a seconda della posizione $\boldsymbol{x}$. Le condizioni inoltre dipendono in generale anche dal tempo è quindi possibile imporre condizioni al contorno non stazionarie che cambino durante la simulazione. Per quanto riguarda le condizioni iniziali esse vanno definite nel file *pCNST.f90* e, nel caso analizzato, sono di pressione e velocità nulle.

Imposizione forzante

Per aggiungere una forza di volume nell'equazione della quantità di moto è necessario utilizzare la funzione *compute_source* in *solution.f90*:

```
1  SUBROUTINE  compute_source(source,mesh,t)
2     USE types
3     USE mesh_mod
4     USE toolbox
5     USE data
6     IMPLICIT NONE
7     TYPE(vector_bloc), INTENT(INOUT) :: source
8     TYPE(mesh_types),  INTENT(IN)    :: mesh
9     REAL(KIND=8),      INTENT(IN)    :: t
10    INTEGER                          :: i, j, k, n, nn, dim, coords(3),&
11                                        dir, dir2, dir3
12    REAL(KIND=8)                     :: x, y, z, nu, Re
13    dim = mesh%dim
14    dir = source%direction
15    dir2 = MOD(dir,dim)+1
16    nu = param%nu
17    Re = 1.d0/param%nu
18    IF (dim==2) THEN
19       IF (dir==1) THEN
20          DO i = 1, mesh%nint(dir2)+1
21             coords(2) = (i-1)*mesh%weights(dir,2)
22             DO j = 1, mesh%nint(dir)+1
23                n = coords(2)+j
24                source%f1(n) = 0d0
25             END DO
26          END DO
27       ELSE IF (dir==2) THEN
28          DO i = 1, mesh%nint(dir2)+1
29             coords(1) = (i-1)*mesh%weights(dir,2)
30             DO j = 1, mesh%nint(dir)+1
31                n = coords(1)+j
32                source%f1(n) = 0d0
```

```
33              END DO
34           END DO
35        END IF
36
37     ELSE
38
39        dir3 = MOD(dir2,dim)+1
40        IF (dir==1) THEN
41           DO k = 1, mesh%nint(dir3)+1
42              coords(3) = (k-1)*mesh%weights(dir,3)
43              DO i = 1, mesh%nint(dir2)+1
44                 coords(2) = (i-1)*mesh%weights(dir,2)
45                 DO j = 1, mesh%nint(dir)+1
46                    n = coords(3)+coords(2)+j
47                    source%f1(n) = 0.0d0
48                 END DO
49              END DO
50           END DO
51        ELSE IF (dir==2) THEN
52           DO k = 1, mesh%nint(dir3)+1
53              coords(3) = (k-1)*mesh%weights(dir,3)
54              DO i = 1, mesh%nint(dir2)+1
55                 coords(2) = (i-1)*mesh%weights(dir,2)
56                 DO j = 1, mesh%nint(dir)+1
57                    n = coords(3)+coords(2)+j
58                    source%f1(n) = 0.0d0
59                 END DO
60              END DO
61           END DO
62        ELSE
63           DO k = 1, mesh%nint(dir3)+1
64              coords(3) = (k-1)*mesh%weights(dir,3)
65              DO i = 1, mesh%nint(dir2)+1
66                 coords(2) = (i-1)*mesh%weights(dir,2)
67                 DO j = 1, mesh%nint(dir)+1
68                    n = coords(3)+coords(2)+j
69                    source%f1(n) = Source_w( (/x,y,z/),t ) ! Da definire
    funzione Source_w
70                 END DO
71              END DO
72           END DO
73        END IF
74
75     END IF
76  END SUBROUTINE compute_source
77
78
79
80  FUNCTION Source_w( x,t )
81     IMPLICIT NONE
82     REAL(KIND=8), DIMENSION(3),  INTENT(IN)  :: x
83     REAL(KIND=8),                INTENT(IN)  :: t
84     REAL(KIND=8)                             :: Source_w
85     REAL(KIND=8)                             :: k, xc, yc, r, z_min, z_max
86
87     k = 350000.0d0
88     xc = 175.0d0
89     yc = 180.0d0
90     r = 10.5d0
```

```
91    z_min = 310.0d0
92    z_max = 350.0d0
93    IF ((x(1) - xc)**2 + (x(2) - yc)**2 < r**2 .AND. x(3) <= z_max .AND. x
      (3) >= z_min) THEN
94       Source_w = k
95    ELSE
96       Source_w = 0.0d0
97    END IF
98 END FUNCTION Source_w
```

dove l'intensità della forza è regolata dal coefficiente k che è stato ricavato inizialmente da una stima delle perdite di carico su casi precedentemente risolti e poi aggiustato in base ai risultati ottenuti per ottenere la velocità desiderata in trachea.

5.2.5 Visualizzazione dei risultati

Il software esporta i risultati in formato vtk (Visualization Tool Kit), in particolare genera quattro file per ogni processore: uno per la pressione e tre per le componenti di velocità. Tali file possono essere aperti utilizzando diversi software di visualizzazione tra i quali si è scelto, allineandosi al gruppo di ricerca, di utilizzare Paraview. Dato il grande numero di test da eseguire e la dimensione contenuta del caso scelto per accelerare la procedura dispendiosa di visualizzazione dei risultati ottenute dai calcoli multiprocessore (con 6 processori già si arriva a 24 file .vtk diversi) si è scelto di implementare nel codice una nuova funzione che esporta i risultati in .pvts. Tale formato appartenente sempre al Visualization Tool Kit che consente di memorizzare i dati in un solo file .vts per processore più un file di controllo .pvts che serve a Paraview per ricostruire il dominio effettivo. In termini di efficienza del codice risulta migliore la scelta di salvare i file separatamente tuttavia la funzione implementata, dettagliata in appendice B, lascia invariati i risultati del lavoro non influendo sui tempi di calcolo e permette di risparmiare tempo in fase di elaborazione dati.

5.3 Set-up della simulazione su OpenFOAM

5.3.1 Generalità sul Software

Per un corretto utilizzo di OpenFOAM è necessario prima di tutto comprenderne l'organizzazione gerarchica: ogni caso standard deve essere definito in una directory principale, in genere con il nome del caso, contenuta nella cartella $RUN dell'installazione di FOAM. Si noti che tutte le utility messe a disposizione da questo strumento sono invece memorizzate nella cartella di installazione nella quale è necessario andare qualora si vogia modificare il sorgente di qualche routine (per esempio solutori). Per lanciare tutti i comandi che la libreria mette a disposizione per il proprio caso è necessario trovarsi nella corrispondente cartella principale e aver definito i parametri necessari all'esecuzione della routine chiamata in file chiamati *dizionari*, tipicamente ogni routine ha bisogno di uno o mpiù dizionari per funzionare. All'interno della cartella principale del caso si trovano sempre altre tre cartelle fondamentali:

- *0*: è la cartella che contiene le condizioni iniziali della simulazione per tutte le variabili coinvolte nel calcolo. Si noti che in generale il solutore può partire da un diverso istante temporale e in quel caso è necessario indicarlo nel file *fvSolution* e

fornire una cartella il cui nome è l'istante temporale di partenza e che contenga i campi al tempo indicato.

- *constant*: contiene le informazioni sulla mesh e sulle proprietà del calcolo che rimangono costanti durante la simulazione, in particolare contiene la cartella *polyMesh* nella quale sono contenute le informazioni della mesh utilizzate dal solutore, la cartella *triSurface* che contiene l'.stl da utilizzare per creare la mesh e altre eventuali cartelle di supporto alla creazione della mesh. Inoltre contiene i file fondamentali *transportProperties* e *turbolenceProperties* che contengono rispettivamente informazioni sul fluido (nel nostro caso Newtoniano con valori di densità e viscosità dell'aria standard) e sulle proprietà turbolente che per motivi illustrati più avanti sono state trattate formalmente con un algoritmo LES.
- *system*: contiene tutti i dizionari fondamentali che verranno presentati in seguito, per il solutore e per l'esecuzione di quasi tutte le routine utilizzate

5.3.2 Generazione della mesh

La generazione della mesh può avvenire con software dedicati e poi inserita, nel formato corretto, nella cartella *constant/polymesh* per essere utilizzata dal solutore, tuttavia OpenFOAM mette anche a disposizione due utility: *blockMesh* e *snappyHexMesh* che consentono nella maggior parte dei casi di ottenere risultati soddisfacenti. La cosa principale che è necessario tenere presente di tali utility è che mentre nei software tradizionali la mesh parte dalla geometria fornita e da li viene estesa fino a raggiungere i bordi del dominio nelle utility di FOAM e in particolare utilizzando *snappyHexMesh* la mesh viene generata a partire da una mesh di volume del dominio completo che viene "scavata" in prossimità della geometria e quindi il risultato finale è ottenuto come una specie di scultura. Tale processo inverso applicato per superfici molto complesse può portare a celle nell'intorno della superficie (che sono quelle tipicamente più importanti) di bassa qualità e a un conseguente degrado nella soluzione ottenuta. Tali aspetti possono comunque essere tenuti sotto controllo e un utente esperto può ottenere mesh di ottima qualità anche utilizzando *snappyHexMesh*. Nel caso presentato avendo a disposizione dizionari utilizzati per geometrie molto simili e volendo mantenere il più possibile le operazioni simili a quelle attualmente utilizzate dal gruppo di ricerca si è scelto di proseguire con l'utilizzo dei meshatori messi a disposizione da OpenFOAM.

Blockmesh

Il primo passo è creare una mesh di background dalla quale degli step successivi verrà ricavata la mesh finale, per fare questo si utilizza l'utility *blockMesh* che consente di creare mesh con elementi esaedrici (meglio se tendenti a cubi per semplificare i passaggi successivi **5OUG2017**) composte da blocchi esaedrici. Da notare che questo step non prende in considerazione l'stl ma crea solo un blocco di partenza della forma e delle dimensioni indicate. Per ottenere questo risultato è sufficiente eseguire il comando *blockMesh* nella directory principale dopo aver opportunamente compilato e inserito nella cartella system il dizionario *blockMeshDict*. Il dizionario utilizzato in questo lavoro è:

```
1 /*--------------------------------*- C++ -*----------------------------*\
2 | =========                 |                                         |
```

```
| \\      /  F ield          | OpenFOAM: The Open Source CFD Toolbox    |
|  \\    /   O peration      | Version:  5                              |
|   \\  /    A nd            | Web:      www.OpenFOAM.org               |
|    \\/     M anipulation   |                                          |
\*-------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
   (0 0 0)
   (0.350 0 0)
   (0.350 0.450 0)
   (0 0.450 0)
   (0 0 0.500)
   (0.350 0 0.500)
   (0.350 0.450 0.500)
   (0 0.450 0.500)


);



blocks
(
   hex (0 1 2 3 4 5 6 7)
   (86 111 123)
   simpleGrading (1 1 1)
);

edges
(
);

boundary
(
   top
   {
      type wall;
      faces
      (
         (3 2 1 0)
      );
   }
   bot
   {
      type wall;
      faces
      (
         (4 5 6 7)
```

```
62          );
63      }
64      right
65      {
66          type wall;
67          faces
68          (
69              (0 4 7 3)
70          );
71      }
72      left
73      {
74          type wall;
75          faces
76          (
77              (1 2 6 5)
78          );
79      }
80      front
81      {
82          type wall;
83          faces
84          (
85              (2 3 7 6)
86          );
87      }
88      back
89      {
90          type wall;
91          faces
92          (
93              (0 1 5 4)
94          );
95      }
96
97  );
98
99  // ************************************************************************//
```

Il risultato ottenuto è mostrato in figura 5.5

SnappyHexMesh

L'utilità *snappyHexMesh* è quella che si occupa di modificare la mesh di partenza affinché prenda la forma della geometria di interesse, rispetto a *blockMesh* le sue funzionalità e opzioni sono molto più estese il che risulta in una complessità molto superiore soprattutto nel saper scegliere i parametri corretti per generare una buona mesh. Per una trattazione esaustiva delle potenzialità di questa routine si rimanda a **5OUG2017**, in questa sezione verranno solo presentati gli step principali che portano alla mesh finale. Per funzionare oltre al dizionario *snappyHexMeshDict* nella cartella system è necessario che nella cartella polymesh sia presente il risultato di *blockMesh* al quale verranno applicati i seguenti step rappresentati in figura 5.6:

- **Castellated**: la prima fase consente di dividere gli esaedri in esaedri più piccoli per rifinire zone di interesse, dopo di che le celle esterne al dominio fluido di

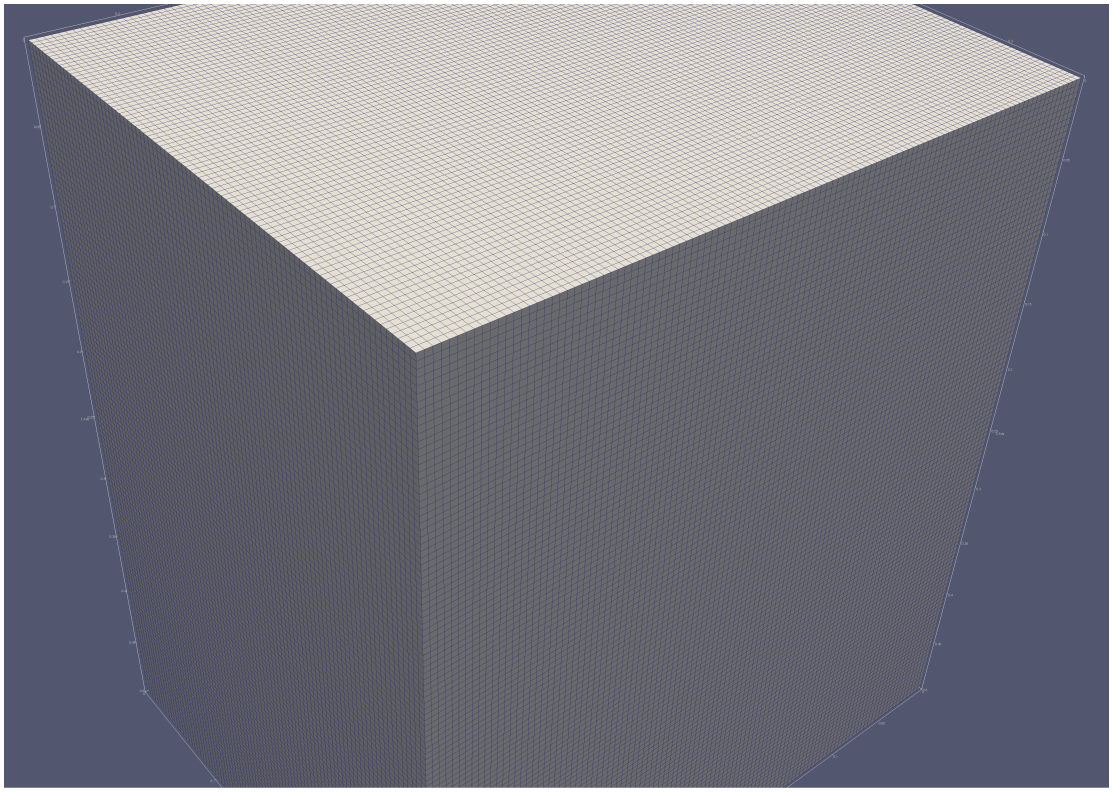


Figura 5.5: *Mesh iniziale esaedrica ottenuta con blockMesh*

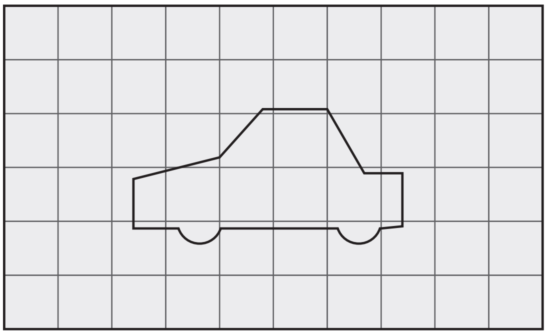
interesse vengono rimosse. Questo da origine a una mesh in cui i bordi della geometria sono approssimati a gradini il che rende le superfici, sopratutto quelle curvilinee non adatte a simulazioni CFD. Alla fine di questa fase comunque la mesh, anche se in modo grezzo tiene conto della geometria inserita.

- **Snap**: La seconda fase consiste nello spostare i vertici degli esaedri per portarli sulla superficie degli STL, risolvere le iterazioni di rilassamento della mesh interna, trovare i vertici che violano i parametri di qualità specificati, ridurre lo spostamento per quei vertici che causano i problemi. La procedura è ripetuta finché la mesh non rispetta i parametri di qualità desiderati

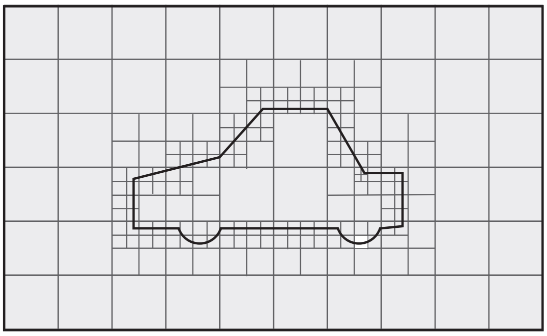
- **Layers**: le zone vicine alle pareti della geometria vengono infittite con strati di celle per aumentare la qualità delle celle risultanti dallo snap e rappresentare meglio lo strato limite. Quest'ultima fase in questo lavoro come in quelli precedenti su geometrie simili non è stato utilizzato poiché la mesh ottenuta con gli step precedenti è soddisfacente.

Il dizionario *snappyHexMeshDict* utilizzato per ottenere la mesh di questo lavoro è:

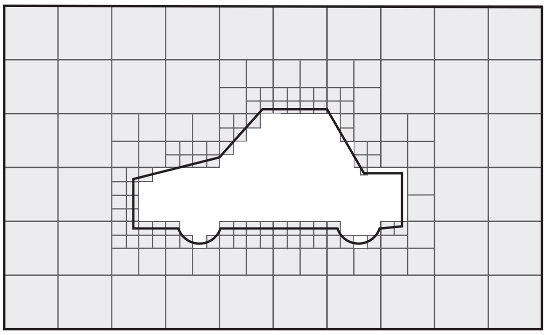
```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                           |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox     |
|  \\    /   O peration     | Version:  5                               |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                |
```



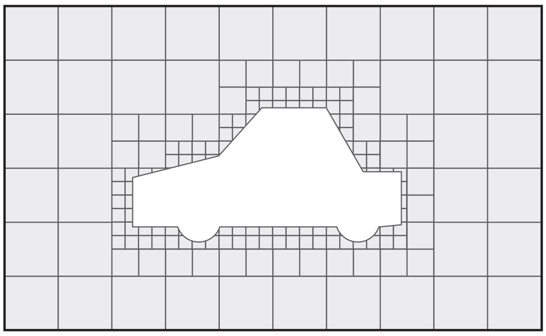
(a) blockMesh



(b) castellated mesh (rifinitura)



(c) castellated mesh (eliminazione)



(d) snap della mesh

Figura 5.6: *Step successivi applicati da snappyHexMesh*

```
|    \\/      M anipulation   |                                              |
\*---------------------------------------------------------------------------*/
FoamFile
{
version      2.0;
format       ascii;
class        dictionary;
object       snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   * * //

#includeEtc "caseDicts/mesh/generation/snappyHexMeshDict.cfg"

castellatedMesh on;
snap            on;
addLayers       off;

geometry //vengono definite le geometrie da includere
{
   CAD // nome della geometria
   {
      type triSurfaceMesh; //tipo geometria
      file "LugaresiFullNewRmm.stl"; // file della geometria
   }

   refinementBox1 // scatola che racchiude la testa
```

```
32      {
33          type searchableBox;
34          min (0.06 0.1 0.05); //estremi della box
35          max (0.25 0.3 0.35);
36      }
37
38  };
39
40  castellatedMeshControls
41  {
42      features // aiuta ad aumentare la qualita' della mesh grazie al
43               // riconoscimento degli spligoli dell'stl
44      (
45          { file "LugaresiFullNewRmm.eMesh"; level 2; }
46      );
47
48      maxLocalCells 1500000; // numero massimo celle per processore durante
        il raffinamento
49      maxGlobalCells 10000000; // numero massimo celle globale durante il
        raffinamento
50      minRefinementCells 1; //se il numero di celle da raffinare e' minore
        di questo numero qui allora il processo si stoppa
51      nCellsBetweenLevels 4; //numero di celle che metti fra un livello e l'
        altro. livello 0 una cella, livello 1 stesso cella divisa in 4
52
53      refinementSurfaces
54      {
55          CAD
56          {
57              level (3 7);
58          }
59      }
60
61      resolveFeatureAngle 30;
62
63      refinementRegions
64      {
65          refinementBox1
66          {
67              mode inside;
68              levels ((1E15 2));
69          }
70      }
71
72      locationInMesh (0.175 0.180 0.350); // punto interno al dominio fluido
73      allowFreeStandingZoneFaces false;
74  }
75
76  snapControls
77  {
78      nSmoothPatch 2;// numero di iterazioni prima di trovare corrispondenza
          tra le superfici
79      tolerance 2.0;// rapporto tra la distanza a cui i putni vengono
        annessi alla superficie e la lunghezza max del bordo(localmente)
80      nSolveIter 50;// numero di iterazioni interne per rendere la mesh dopo
          gli spostamenti delle celle migliore
81      nRelaxIter 10;//numero massimo di interazioni prima di rilassare la
        qualita' richiesta
82  }
```

```
83
84 meshQualityControls//parametri di qualita' della mesh
85 {
86     maxNonOrtho 50;
87     maxBoundarySkewness 6;
88     maxInternalSkewness 4;
89     maxConcave 80;
90     minVol -1e-30;
91     minTetQuality 1e-15;
92     minArea 1e-13;
93     minTwist 0.05;
94     minDeterminant 0.001;
95     minFaceWeight 0.05;
96     minVolRatio 0.01;
97     minTriangleTwist -1;
98     nSmoothScale 4;
99     errorReduction 0.75;
100 }
101
102 mergeTolerance 1e-6;
103
104 //
      ************************************************************************
       //
105
```

I principali parametri da indicare per la fase di castellazione sono:

- *maxLocalCells*: massimo numero di celle per processore durante il raffinamento
- *maxGlobalCells*: limite di celle globale durante il raffinamento
- *minRefinementCells*: numero minimo di celle da raffinare;
- *nCellsBetweenLevels*: numero di celle di "buffer" tra diversi livelli di raffinamento;
- *refinementSurfaces*: subdizionario che specifica come raffinare le superfici;
- *resolveFeatureAngle*: applica il massimo livello di raffinamento alle celle, il quale angolo di intersezione è maggiore di quello indicato;
- *locationInMesh*: Coordinate di un punto che specifica l'interno della mesh, la celle esterne al volume desiderato verranno scartate.

quelli per la fase di snappy:

- *nSmoothPatch*: numero di iterazioni della levigatura prima di trovare corrispondenza tra le superfici;
- *tolerance*: rapporto tra distanza massima dalla quale il vertice può essere attratto sulla superficie, rispetto la distanza massima locale tra due punti della mesh;
- *nSolveIter*: numero di iterazioni di rilassamento della mesh;
- *nRelaxIter*: numero di iterazioni di snappy.

mentre i principali criteri di qualità sono:

- *maxNonOrtho*: massima non ortogonalità consentita;
- *maxConcave*: massima concavità permessa;
- *minTetQuality*: minima qualità dei tetraedri permessa;
- *minArea*: minima superficie delle facce concessa;
- *minDeterminant*: minimo valore del determinante normalizzato di una cella permesso.

La creazione della mesh può richiedere parecchio tempo a seconda del numero di celle, per questo motivo snappyHexMesh può essere lanciato in parallelo usando MPI. Per farlo è necessario dalla cartella principale lanciare il comando *decomposePar* che si occupa di decomporre il dominio secondo le istruzioni date nel dizionario *decomposeParDict* successivamente si lancia snappy con il comando

```
mpirun -np N -parallel snappyHexMesh -overwrite
```

dove il flag *-overwrite* è utile per mantenere in memoria solo la mesh finale. Una volta conclusa la creazione della mesh essa può essere ricomposta utilizzando il comando

```
reconstructParMesh -latestTime
```

La mesh ottenuta e utilizzata per la soluzione con OpenFOAM per questo lavoro risulta di 18 904 202 celle che è un numero elevato, per esempio la tesi **5LM2014** utilizzava circa dieci milioni di celle su una geometria analoga. La differenza è da attribuirsi a due cause principali: la prima è che seppure la geometria in esame è simile nel caso presentato la mesh contiene tutto il volume della scatola in cui è immersa la testa per avere un caso il più simile possibile a quello simulato con il codice di Guermond la seconda è che in questo lavoro la simulazione che si vuole ottenere è una DNS e quindi è necessario che le celle siano sufficientemente piccola da essere rappresentative per tutte le scale turbolente. Un dettaglio della mesh finale è mostrato in figura 5.7

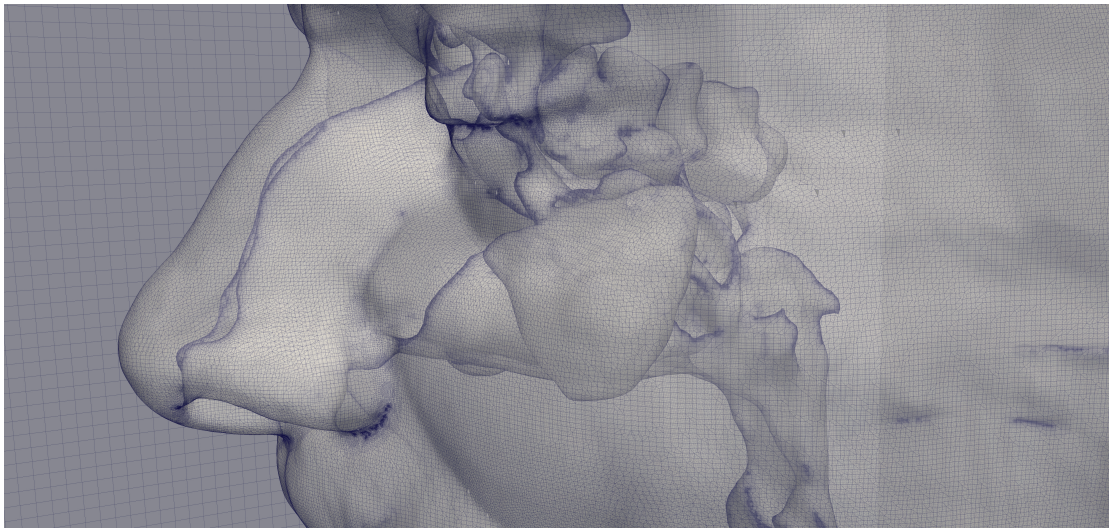


Figura 5.7: *Dettaglio della mesh finale*

5.3.3 Modello di turbolenza

Seppure la simulazione che si vuole ottenere è una DNS formalmente viene eseguita come una LES per avere la certezza che la dimensione delle celle sia tale da risolvere tutte le scale turbolente. Infatti utilizzando una simulazione LES con una griglia sufficientemente fine implica disattivare il modello di turbolenza ed eseguire in pratica una DNS. Poiché quindi il modello LES rappresenta solo un check per verificare che la dimensione delle celle sia opportuna non si riportano gli aspetti teorici e modellistici ma solamente l'implementazione in OpenFOAM, per maggiori dettagli su questo tipo di simulazioni si rimanda a **5P2000**. Il modello utilizzato è WALE (Wall-Adapting Locel Eddy-viscosity) per i cui dettagli sono discussi in **5FD1999** e l'implementazione avviene attraverso il dizionario *turbulenceProperties* nella cartella *constant* che, in questo caso è:

```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                            |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox      |
|  \\    /   O peration     | Version:  5                                |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                 |
|    \\/     M anipulation  |                                            |
\*---------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
location    "constant";
object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType LES;

LES
{

   LESModel        WALE;
   turbulence      on;
   printCoeffs     on;
   delta           cubeRootVol;
   cubeRootVolCoeffs
   {
      deltaCoeff      1;
   }

   vanDriestCoeffs
   {
      delta           cubeRootVol;
      cubeRootVolCoeffs
      {
         deltaCoeff      1;
      }

      smoothCoeffs
      {
         delta           cubeRootVol;
         cubeRootVolCoeffs
```

```
        {

            deltaCoeff      1;
        }

        maxDeltaRatio   1.1;
    }

    Aplus           26;
    Cdelta          0.158;
  }
}
// ************************************************************************* //

```

Dal punto di vista del costo computazionale l'utilizzo del modello nel caso riportato non sembra avere impatto, infatti test condotti disattivando il modello di turbolenza evidenziano che il tempo di soluzione di OpenFOAM risulta sostanzialmente invariato sia per la soluzione seriale che per quella in parallelo.

5.3.4 Condizioni iniziali e al contorno

Per la definizione delle condizioni al contorno è necessario prima di tutto definire le geometrie sulle quali andranno imposte. Nel nostro caso questo è stato fatto per le pareti del dominio definendo le facce come *boundary* nel dizionario *blockMeshDict* mentre per la superficie stl nel dizionario *snappyHexMeshDict* sotto la voce *geometry*. Una volta definite le geometrie le condizioni sia iniziali che al contorno sono specificate in appositi dizionari con il nome della variabile a cui si riferiscono posizionati nella cartella *0*. Nel caso in esame sono necessari tre file: uno per la pressione, uno per la velocità e uno per la viscosità del modello della turbolenza utilizzato ν_{sgs}. Le condizioni al contorno da imporre sono quelle di parete solida per tutte le superfici del dominio incluso l'stl, quindi nulle la velocità e il gradiente di pressione mentre come condizioni iniziali, come nel codice di Guermond, sono stati imposti campi nulli. Per quanto riguarda la viscosità turbolenta è stato imposto gradiente nullo su tutte le superfici e una condizione iniziale omogenea $\nu_{sgs} = 1$. A titolo di esempio di riporta il file della velocità:

```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       volVectorField;
object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

dimensions      [0 1 -1 0 0 0 0];

```

```
internalField   uniform (0 0 0);

boundaryField
{
    top
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }bot
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    front
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    back
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    right
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    left
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
}
// ************************************************************************//


```

Imposizione forzante

Per imporre la forza di volume è necessrio modificare in qualche modo l'equaizone della quantità di moto agiungendo il termine voluto, a seconda del solutore utilizzato questo può essere fatto modificasndo direttamete il temrine sorgente o utilizzando classi già predefinite da OpenFOAM. Quest'ultimo è il nostro caso dato cheil solutore permette di definire nel dizionario *fvOptions* posizionato nella cartella constant il temrine forzante dell'equaizone. l'implementazione prevede due passaggi:

- definizione del volume nel quale si vuole inserire la forza
- definizione della forza da introdurre

Per la definizione del volume si è utilizzata l'utility *topoSet* controllata dal dizionario *topoSetDict* da posizionare nella cartella *system*. Nel caso in esame la struttura di tale dizionario è:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
object      topoSetDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

actions
(
{
name    forza;
type    cellSet;
action  new;
source  cylinderToCell;
sourceInfo
{
p1 (0.175 0.180 0.310);
p2 (0.175 0.180 0.350);
radius 0.009;
}
}
);

// ************************************************************************* //
```

che definisce un set di celle di nome *forza* contenente tutte quelle comprese nel cilindro definito dalle posizioni dei centri delle due facce circolari e dal raggio. Notare che per questo set va aggiunto alla mesh finale utilizzando, dopo aver eseguito snappy, il comando:

```
topoSet
```

A questo punto per imporre la forzante è necessario definire il dizioanrio *fvOptions* nella cartella *constant*, nel nostro caso tale file risulta:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
object      topoSetDict;
```

```
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

momentumSource
{
   type            meanVelocityForce;
   active          yes;

   meanVelocityForceCoeffs
   {
      selectionMode   cellSet;
      cellSet       forza;

      fields      (U);
      Ubar        (0 0 1.1);
   }
);

// ************************************************************************* //
```

Il tipo di forzante scelto *meanVelocityForce* consente di specificare la velocità media desiderata nel volume e applica la forza necessaria per ottenerla che è esattamente il tipo di forzante che si vuole imporre in questo caso. Come per il codice di Guermond la velocità target in trachea è $1.1\,m/s$ diretta come l'asse z

5.3.5 Solutore

Tra i solutori disponibili presentati nel capitolo 4 quello utilizzato è PimpleFOAM che, a causa della mancanza di problematiche di convergenza, è utilizzato in modalità PisoFOAM ovvero con una sola sotto iterazione e due correzioni ad ogni passo temprale. Il solutore IcoFOAM potrebbe essere utilizzato poiché l'obiettivo è un calcolo DNS tuttavia per essere sicuri di avere una mesh sufficiente fine il metodo utilizzato è eseguire una LES e verificare che il modello di turbolenza abbia avuto impatto nullo sulla soluzione ottenuta. SimpleFOAM è escluso poiché le simulazioni effettuate non sono stazionarie e per un'implementazione instazionaria occorrerebbe eliminare il rilassamento compromettendo la stabilità del metodo. Inoltre vari lavori **4B1998**, **4ML1993**, **4WS1989**, **4JJA1986** concludono che la strategia PISO è più efficiente e ha proprietà di convergenza migliori nella soluzione per problemi transitori.
Le impostazioni del solutore sono contenute in due dizionari contenuti nella cartella *system*. Il primo è *fvSchemes* in cui sono contenuti gli schemi di approssimazione dei vari termini delle equazioni da risolvere, in questo caso si è scelto di mantenere le scelte effettuate in precedenza dal gruppo di lavoro e descritte in **5LM2014** indicate nel dizionario:

```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
```

```
{
version     2.0;
format      ascii;
class       dictionary;
object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

ddtSchemes
{
default          backward;
}

gradSchemes
{
default          Gauss linear;
grad(p)          Gauss linear;
grad(U)          Gauss linear;
}

divSchemes
{
default          none;
div(phi,U)       Gauss linear;
div(phi,R)       Gauss linear;
div(R)           Gauss linear;
div((nuEff*dev(T(grad(U))))) Gauss linear;
div((nuEff*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
default          Gauss linear corrected;
}

interpolationSchemes
{
default          linear;
interpolate(U)   linear;
}

snGradSchemes
{
default          corrected;
}

fluxRequired
{
default          no;
p                ;
}

// *********************************************************************//
```

Il secondo dizionario necessario è *fvSolution* nel quale sono definiti i parametri di controllo del solutore che sono stati presentati nel capitolo 4.

```
/*--------------------------------*- C++ -*----------------------------*\
```

```
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    * * //

solvers
{
p
{
solver           GAMG;  //PCG
tolerance        1e-7;
relTol           0.01;

smoother         DICGaussSeidel;

cacheAgglomeration true;
nCellsInCoarsestLevel 10;
agglomerator     faceAreaPair;
mergeLevels      1;
}

pFinal
{
$p;
relTol          0;
}

"(U|k|epsilon)"
{
solver          smoothSolver;
smoother        symGaussSeidel;
tolerance       1e-05;
relTol          0.1;
}

"(U|k|epsilon)Final"
{
$U;
relTol          0;
}
}

PIMPLE
{
nOuterCorrectors    1; //correzioni esterne
nNonOrthogonalCorrectors 0;
nCorrectors         2; //correzioni interne
}
```

```
// ************************************************************************//
```

In particolare nel file riconosciamo le indicazioni dei solutori da utilizzare per le equazioni della pressione e quantità di moto, il numero di correzioni dell'algoritmo PISO impostato a due e il numero di sotto iterazioni del PIMPLE settato appunto a uno per far lavorare l'algoritmo come un PISO.

5.3.6 Controllo della simulazione

Il controllo dei parametri fondamentali della simulazione è effettuato mediante il dizionario *controlDict* contenuto nella cartella *system*. Il file utilizzato nel nostro caso è

```
/*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     pimpleFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         0.001;

deltaT          1e-5;

writeControl    runTime;

writeInterval   2e-4;// intervallo di tempo salvataggio soluzione

purgeWrite      0;

writeFormat     ascii;

writePrecision   8;

writeCompression off;

timeFormat      general;

timePrecision   6;
```

```
44
45 runTimeModifiable true;
46
47 adjustpasso temprale    no;
48
49 // ************************************************************//
```

dove si nota lo stesso passo temporale e tempo finale, per un totale di cento iterazioni, impostato anche nel solutore di Guermond.

5.3.7 Visualizzazione dei risultati

OpenFOAM è nativamente legato al software di visualizzazione ParaView già utilizzato dal gruppo di ricerca e al quale standard si è adeguata anche la visualizzazione dei risultati del solutore di Guermond. L'integrazione tra i due software è tale che utilizzando il comando

```
paraFoam
```

nella cartella principale del caso esso verrà completamente caricato e aperto in ParaView con mesh, risultati e tutto ciò che è stato definito.

CAPITOLO 6

Presentazione dei risultati ottenuti

In questo capitolo vengono presentati i risultati principali ottenuti con entrambi i codici utilizzati soffermandosi su quelli che sono gli aspetti più rilevanti e significativi per l'indagine svolta. Particolare attenzione è posta al confronto di efficienza di calcolo delle due strategie e ai limiti di validità di quanto ottenuto.

6.1 Risultati del codice DNS

La soluzione ottenuta con il solutore DNS ha rivelato un problema nel codice e in particolare nell'implementazione del metodo dei contorni immersi.
Come mostrato in figura 6.1 e 6.2 la superficie immersa risulta porosa e la maggior parte della portata non rimane confinata della cavità nasale ma esce attraverso le pareti. Test su geometrie molto più semplici hanno confermato che il problema non era nella complessità dell'stl utilizzato ma intrinseco nella procedura implementata. Per questo motivo si è scelto di non sprecare risorse nel tentativo di sistemare il codice poiché:

- il problema riscontrato non compromette drammaticamente l'obiettivo del lavoro presentato che non è quello di ottenere una soluzione corretta con il software in questione ma di indagare quantitativamente lo speedup della procedura che è possibile ottenere sostituendo il solutore attualmente usato. Infatti Il codice completa la sua esecuzione e converge a una soluzione anche se questa non è fisica non rispettando le condizioni al contorno sulla geometria immersa. É comunque importante sottolineare che il ciclo del metodo dei contorni immersi implementato viene eseguito (come si può intuire anche dal campo di moto in figura 6.1) anche se sembra non avere abbastanza autorità per imporre completamente le condizioni richieste. Tale aspetto è fondamentale poiché, unito al fatto che l'algoritmo presentato è validato e testato, garantisce che il tempo di calcolo del solutore tiene

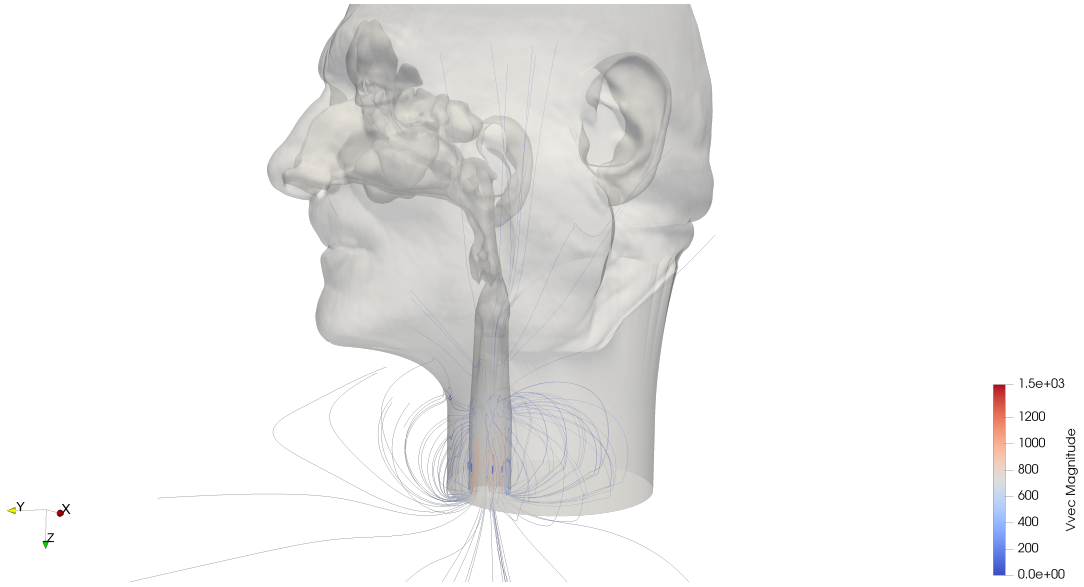


Figura 6.1: *Streamline passanti per la trachea ottenute dalla soluzione del codice DNS.*

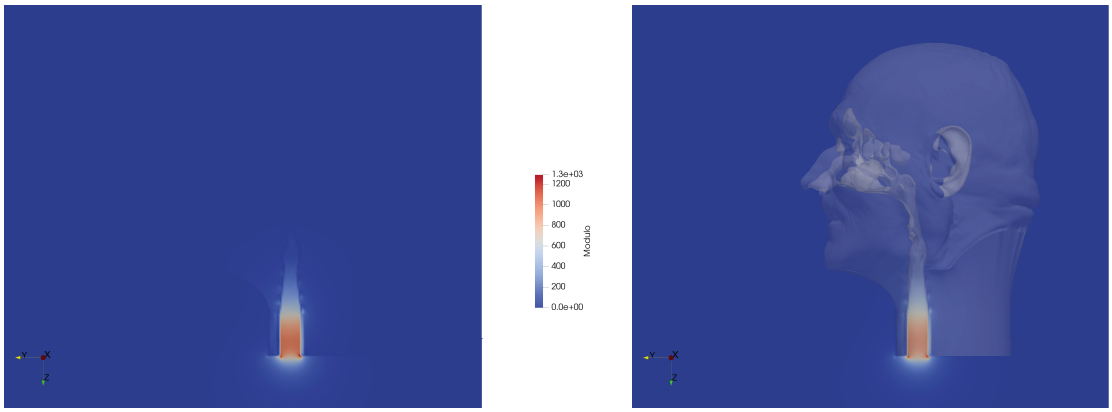


Figura 6.2: *Sezione sagittale del campo di moto ottenuta con il solutore DNS. A destra si è sovrapposta la geometria inserita.*

conto anche del metodo dei contorni immersi e solo lievi variazioni potrebbero nascere sistemando il problema puramente di natura implementativa.

- dopo aver condotto un'indagine esplorativa e alcuni test preliminari è risultato impossibile determinare con sufficiente sicurezza le tempistiche di soluzione del problema incontrato

- lo stesso gruppo di ricerca non ha interesse nel funzionamento del codice in questione poiché fin dall'impostazione di questo lavoro era stato identificato come candidato alla sostituzione dell'attuale solutore di OpenFOAM un programma potenzialmente più veloce e moderno di quello utilizzato che era in corso di sviluppo contemporaneamente a questo lavoro.

6.2 Risultati con OpenFOAM

La soluzione ottenuta con OpenFOAM risulta qualitativamente in linea con i risultati di precedenti simulazioni del gruppo di ricerca. Con riferimento al campo di moto di una sezione sagittale mostrato in figura 6.3 il fluido risulta sostanzialmente in quiete nella maggior parte del dominio che risulta abbastanza esteso da evitare condotti esterni fittizi di connessione tra le narici e la gola.

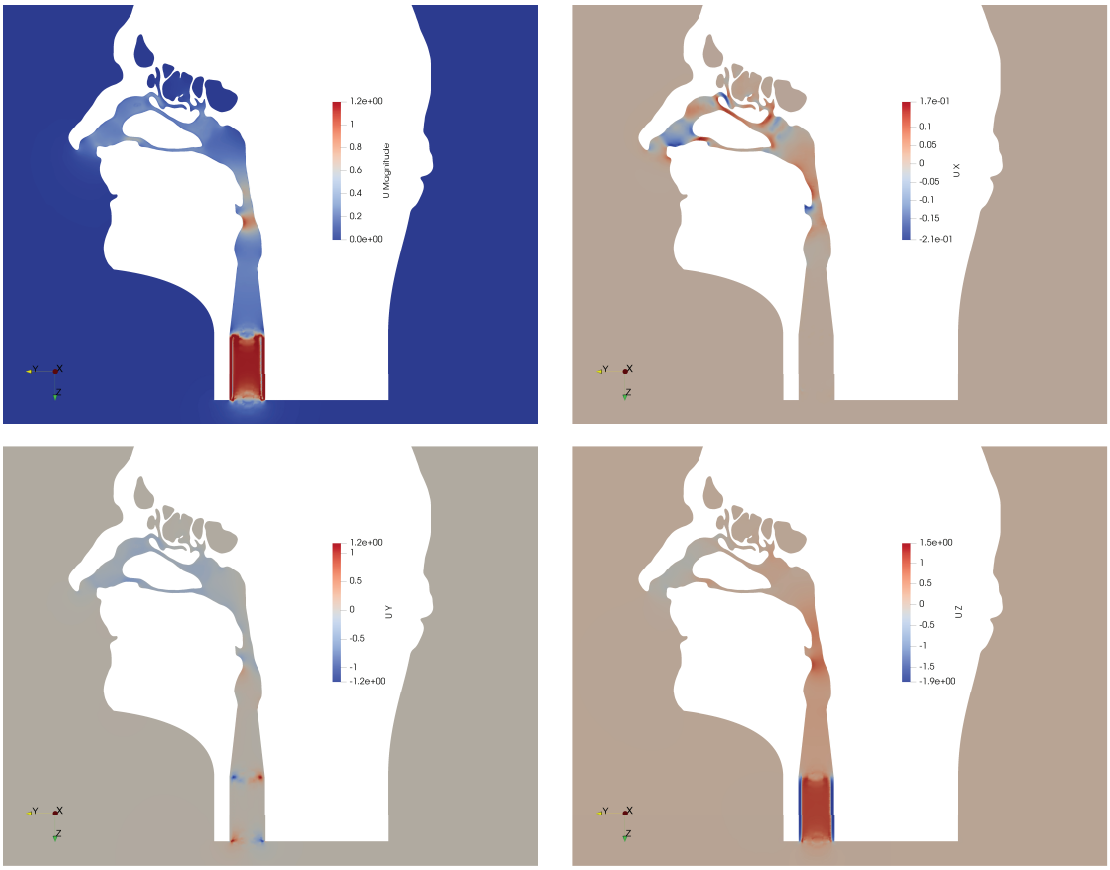


Figura 6.3: *Sezione sagittale del campo di velocità ottenuto con OpenFOAM. Il modulo della velocità è rappresentato in alto a sinistra e di seguito le tre componenti U_x, U_y e U_z*

L'aria viene introdotta nelle due cavità attraverso le narici che ne provocano una prima accelerazione nella direzione assiale del condotto che risulta, fino alle valvole nasali, circa allineato con l'asse z. Il flusso si incanala poi nei vari meati accelerando a causa della riduzione di sezione, per poi rallentare in corrispondenza delle coane. Parte dell'aria entra nei seni paranasali, con una velocità generalmente molto bassa. In corrispondenza della nasofaringe il flusso ruota di circa 90° per allinearsi con l'orofaringe provocando una zona di ricircolo nella parte superiore della zona. Il flusso precedentemente suddiviso nelle due fosse nasali, ora ricongiunto, si dirige verso il restringimento dell'epiglottide dove la velocità aumenta e si genera un getto che si estende per tutta la laringofaringe. Gli effetti della forzante beni visibili dal campo di moto rimangono limitati a una zona artificiale senza impattare significativamente sul risultato ottenuto
Per quanto riguarda il campo di pressione mostrato in figura 6.4 l'attrito sulle pareti

delle cavità nasali provoca una graduale caduta di pressione che raggiunge il suo minimo alla fine del condotto.

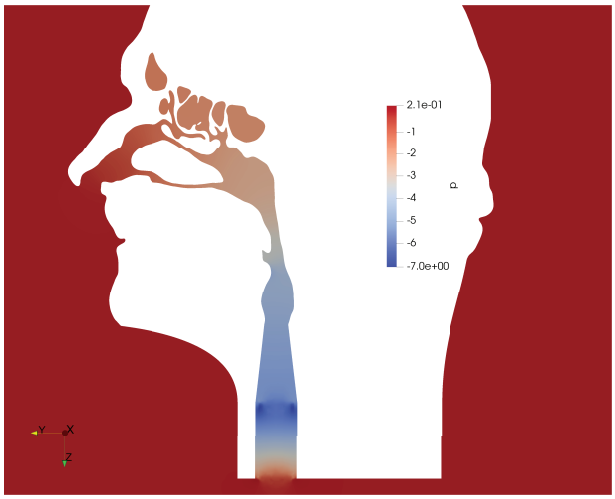


Figura 6.4: *Sezione sagittale del campo di pressione ottenuto con OpenFOAM*

Sono presenti due cadute di pressione principali: la prima, più graduale, nei meati e la seconda, più brusca, in corrispondenza dell'epiglottide dove l'aumento di sezione provoca una rapida espansione del fluido. In 6.5 è rappresentato il campo di moto di una sezione coronale dove che mostra meati e turbinati.
Notare che la corrente nei seni paranasali risulta mediamente stagnate ma con picchi localizzati di velocità. La pressione della stessa sezione mostrata in figura 6.6 evidenzia le cadute di pressione nelle aree di sezione più stretta.
A causa del problema nel codice DNS un confronto tra le due soluzioni ottenute non risulta fattibile ma questo non impatta sugli obiettivi del lavoro per i quali è sufficiente che la simulazione OpenFOAM sia impostata realisticamente rispetto agli standard del gruppo di lavoro e converga a una soluzione fisica per poter considerare significativi i tempi di calcolo. Tale requisito risulta soddisfatto, infatti i risultati ottenuti sono qualitativamente in linea con quelli già disponibili nel gruppo di ricerca (per esempio **5LM2014**), anche se un confronto quantitativo è reso complesso a causa della diversità delle condizioni al contorno e di forzamento imposte.

6.3 Confronto efficienza di calcolo

6.3.1 Caratteristiche hardware

In questa sezione verranno presentati i risultati principali del lavoro che riguardano le prestazioni del codice. I risultati ottenuti, specialmente quelli di scaling parallelo, dipendono dalle caratteristiche hardware della macchina utilizzata e, sopratutto per simulazioni che coinvolgono un gran numero di processori, dall'implementazione delle librerie MPI utilizzata. Per ragioni di flessibilità nell'effettuare i vari test in questo lavoro si è utilizzato un personal computer la cui architettura non è pensata per il calcolo parallelo e la versione general porpouse della libreria OpenMPI. Affinché i risultati di un confronto di questo tipo siano validi è necessario che l'architettura sulla quale vengono eseguiti i due codici sia la stessa tuttavia presentando le specifiche hardware utilizzate

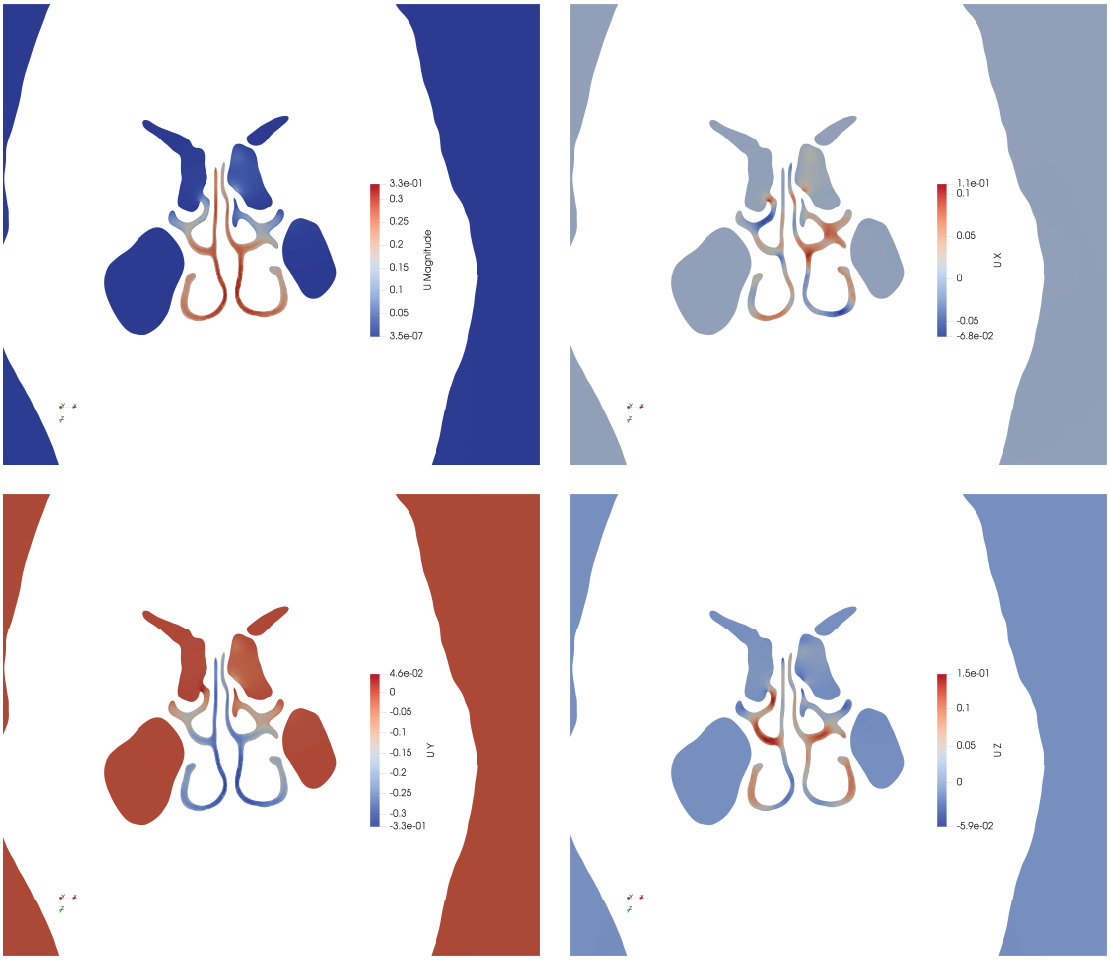


Figura 6.5: *Sezione coronale del campo di velocità ottenuto con OpenFOAM. Il modulo della velocità è rappresentato in alto a sinistra e di seguito le tre componenti U_x, U_y e U_z*

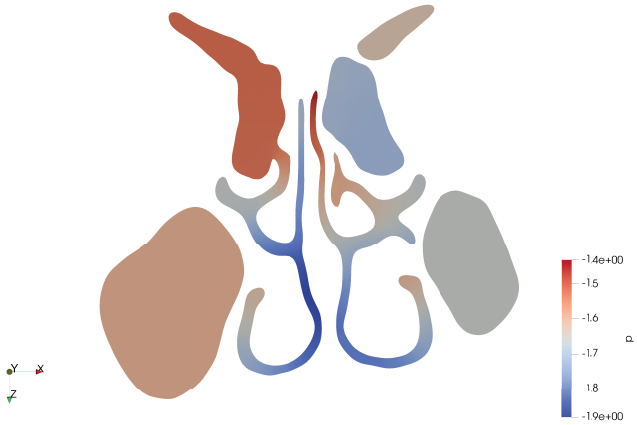


Figura 6.6: *Sezione coronale del campo di pressione ottenuto con OpenFOAM*

i risultati acquisiscono una valenza più generale. Per questo motivo di seguito sono presentate le caratteristiche hardware della macchina utilizzata importanti per il lavoro svolto:

- **Sistema operativo**: Il sistema operativo utilizzato è la versione di linux: Ubuntu 18.04 LTS
- **Scheda madre**: La scheda madre utilizzata è una Asus Rampage IV Extreme dotata di chipset per il microprocessore Intel X79 e con un soket LGA 2011/Soket R. Tale scheda presenta 4 canali di memoria ciascuno con due slot(tecnologia dual channel) e permette l'installazione al massimo di 64 GB di memoria RAM DDR3. La frequenza dei bus di memoria è 2400 MHz
- **Microprocessore**: Il microprocessore del sistema è un Intel® Core™ i7-4930K, una CPU con 6 core, 12 MB di memoria Cache e una frequenza di lavoro di 3.40 GHz
- **Memoria**: Sulla macchina sono installati 48 GB di memoria RAM UDIMM DDR3 in 6 slot(3 canali dual channels) da 8 GB ciascuno, la frequenza dei bus è 1600 MHz e la latenza di classe CL10

6.3.2 Confronto seriale

I risultati più importante di questo lavoro sono sicuramente i tempi di calcolo necessari a ciascuno dei due codici. In tabella 6.1 è riassunto il risultato principale I numeri

	numero celle mesh	tempo di soluzione [s]	speedup
OpenFOAM	18 904 202	28 798	-
Guermond	19 044 480	2 632	**10.94**

Tabella 6.1: *Tempi di calcolo seriali per i due software utilizzati*

presentati per entrambi i casi sono relativi alla sola esecuzione del ciclo temporale di 100 timestep escludendo quindi tutte le operazioni di input, output e preparazione della mesh. Per PimpleFOAM tale numero è prodotto di default alla fine di ciascun passo temporale dell'algoritmo PISO mentre per il codice DNS si è utilizzata la routine $user_time()$. I test sono stati eseguiti mantenendo attivi sulla macchina solo i processi di default del sistema operativo per garantire la minor interferenza possibile.
Il risultato fondamentale ottenuto è lo speedup del codice DNS utilizzato rispetto alla procedura attuale di circa unici volte. Tale numero giustifica completamente tutti gli sforzi necessari per la ricerca e il passaggio dalla procedura attuale a una basata su algoritmi di soluzione più veloci. É comunque necessario sottolineare che questo è il risultato di un analisi preliminare limitata in alcuni aspetti critici che verranno presentati e discussi nel capitolo 7.

6.3.3 Confronto scaling parallelo

I dati fondamentali tipicamente considerati quando si parla di scalabilità di un codice due: la scalabilità forte, ovvero lo speedup del tempo di soluzione di un problema con dimensione fissata all'aumentare del numero di processori utilizzati e quella debole

ottenuta mantenendo costante il numero di punti risolto da ciascun processore e considerando la variazione del tempo di soluzione all'aumentare della dimensione globale del problema dovuta all'aumento del numero di processori utilizzati. In questo caso ci si è concentrati sul primo parametro data la difficoltà di generare mesh soddisfacenti di dimensione simile per i due codici senza esaurire la memoria a disposizione. I risultati sono riportati in tabella 6.2 e rappresentati in figura 6.7. I numeri riportati sono acqui-

	Tempo di soluzione [s]		Speedup parallelo	
processori	OpenFOAM	Codice DNS	OpenFOAM	Codice DNS
1	29 232	2 632	-	-
2	17 899	1 592	1.63	1.65
4	12 191	885	2.40	2.97
6	10 593	665	2.76	3.96
8	12 330	689	2.37	3.82
12	11 414	642	2.56	4.10

Tabella 6.2: *Scaling parallelo per i due software utilizzati rispetto all'esecuzione parallela su core singolo*

siti nello stesso modo presentato nel caso seriale, tuttavia sono relativi per entrambi i codici al processore più lento. Anche in questo caso durante i test non erano presenti altre applicazioni in esecuzione sulla macchina.
I risultati ottenuti si discostano da quelli riportati in letteratura (per esempio da **3GM2011** e **4AR2016**) i quali dichiarano per entrambi i codici scaling circa lineare fino ad almeno 1000 processori. Tale fatto non deve sorprendere, infatti come già accennato, i risultati di scaling parallelo sono strettamente legati all'architettura della macchina utilizzata quindi nell'interpretarli è necessario tenere conto dell'hardware e della versione standard delle librerie OpenMPI utilizzata. I risultati presenti il letteratura al contrario sono riferiti a test progettati ed ottimizzati per il calcolo parallelo. Un prima conclusione è quindi che l'efficienza parallela dei codici in questo lavoro è limitata dall'hardware. Premesso ciò il codice DNS sembra adattarsi meglio alla macchina a disposizione infatti scala consistentemente meglio rispetto ad OpenFOAM. Tale osservazione rende plausibile il risultato teorico descritto dagli autori dell'algoritmo in **3GM2011**, di scaling lineare massivo su architetture appositamente progettate il che potrebbe rappresentare un ulteriore vantaggio per problemi di grandi dimensioni dati gli scarsi risultati testimoniati in letteratura dello scaling massivo di OpenFOAM. Per il gruppo di ricerca tale risultato non rappresenta un vantaggio immediato poiché le simulazioni vengono svolte utilizzando un numero di processori dell'ordine delle decine tuttavia, potrebbe essere significativo qualora la macchina utilizzata non risultasse perfettamente ottimizzata. I numeri presentatati mostrano un crollo di scalabilità oltre i 6 processori che è il numero di cores fisici della macchina utilizzata dopo i quali il tempo di soluzione rimane sostanzialmente costante e in alcuni casi aumenta. Questo perché utilizzando più processori rispetto a quelli disponibili aumentano le comunicazioni necessarie per la soluzione senza che incrementi la potenza di calcolo. Per questo motivo, come noto anche in letteratura, è suggerito utilizzare tutti e soli i processori disponibili.

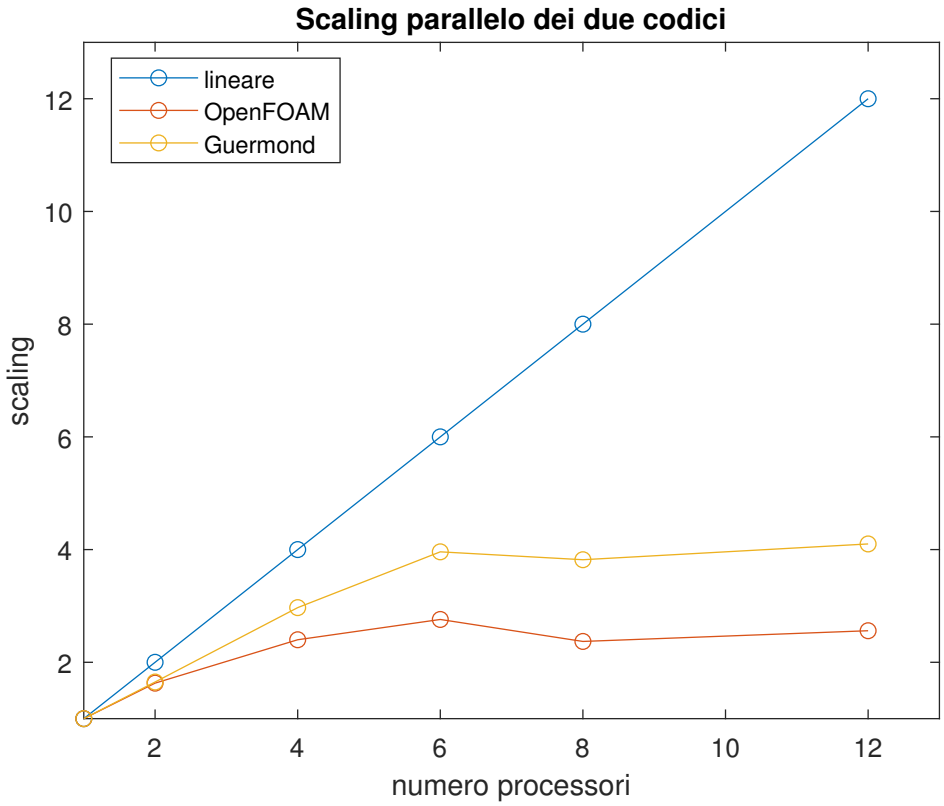


Figura 6.7: *Scaling parallelo forte dei due codici*

CAPITOLO 7

Conclusioni e possibili sviluppi futuri

In questo capitolo sono discussi i risultati ottenuti analizzandoli con particolare attenzione alla collocazione di questo lavoro in un progetto più ampio che porti sperabilmente a un aumento significativo di efficienza nella procedura grazie all'implementazione delle idee, seppur in forma e modalità diverse, illustrate in questo lavoro. In questo senso vengono sottolineati i limiti di validità di quanto ottenuto e indicate le linee guide suggerite per proseguire il lavoro.

7.1 Limiti dei risultati

Per quanto i risultati ottenuti siano soddisfacenti è necessario evidenziarne alcuni limiti. Il primo è sicuramente il fatto che lo speedup ottenuto di circa un fattore 11 è valido per mesh con lo stesso numero di celle, tuttavia il codice presentato e in particolare tutti i solutori basati su griglie parallelepipede devono necessariamente soddisfare delle condizioni stringenti sulla forma del dominio e quindi inevitabilmente aggiungere celle rispetto alla griglia non strutturata di geometria arbitraria che è possibile utilizzare su OpenFOAM.
Un secondo limite già accennato riguarda l'architettura della macchina utilizzata, infatti i risultati ottenuti in termini di scaling parallelo possono solo fornire alcune indicazioni per le futura indagini ma a causa delle caratteristiche hardware del PC utilizzato è impossibile trarre conclusioni sulle reali prestazioni dei due codici.
Un terzo limite che potrebbe incrementare lo speedup è il trattamento delle celle contenute nella geometria solida nell'implementazione del metodo dei contorni immersi. Infatti tali celle in teoria potrebbero non essere calcolate con un risparmio sul tempo di soluzione che può diventare considerevole per geometrie con volumi elevati. Il codice utilizzato poiché ha come obiettivo la possibilità di risolvere problemi di interazione fluido-struttura con grandi spostamenti necessita il calcolo di tute le celle.

7.2 Possibili sviluppi futuri

Visti i limiti incontrati, per lo sviluppo futuro di questo progetto una volta che il nuovo software sarà disponibile e funzionante, si suggerisce:

- Confrontare i tempi di calcolo del nuovo codice e di OpenFOAM a parità di soluzione ottenuta, ovvero ciascun software utilizzando la minima mesh necessaria per ottenere una soluzione corretta
- Per un significativo confronto della scalabilità dei codici in parallelo è necessario l'utilizzo di architetture con più di 1000 core appositamente progettate per questo tipo di calcoli. Infatti per entrambi i codici in letteratura è presente la validazione dello scaling lineare fino ad almeno mille processori, il che vuol dire che un eventuale guadagno si potrebbe ottenere oltre tale soglia. Per gli interessi del gruppo di ricerca che non utilizza macchine di tali dimensioni potrebbe risultare utile confrontare gli scaling dei due codici sull'architettura utilizzata in modo da verificare che garantisca uno scaling lineare per entrambi i codici.
- Valutare, qualora il nuovo codice non lo preveda, l'effetto benefico di un'implementazione ad-hoc del metodo dei contorni immersi che, per i corpi rigidi, sia in grado di evitare il calcolo delle celle interne al volume non fluido.

7.3 Conclusioni

In conclusione per sostituire OpenFOAM nella procedura di calcolo del gruppo di ricerca è necessario testare in modo più approfondito il nuovo codice, tuttavia il risultato di questo lavoro dimostra che lo speedup dei tempi di calcolo ottenibile modificando il solutore giustifica completamente sia il proseguimento della ricerca in questa direzione che il probabile sforzo che sarà necessario per l'aggiornamento effettivo del metodo utilizzato.

APPENDICE $\mathcal{A}$

Test preliminari per l'adattamento del codice

In questa appendice vengono presentati i test preliminari che sono stati condotti per verificare che il codice disponibile fosse in grado di effettuare simulazioni del tipo di interesse per il gruppo di ricerca. Particolare importanza è data alle problematiche riscontrate che hanno portato più volte al cambiamento di rotta di questo lavoro fino a definire un test case che fosse compatibile con entrambi i codici da utilizzare e significativo per i problemi pratici di interesse del gruppo di ricerca.

A.1 Test condizioni al contorno

A.1.1 Obiettivo del test

Come già accennato il codice utilizzato lavora su una griglia cartesiana definita nel parallelepipedo $[L_x, L_y, L_z]$ pertanto le condizioni al contorno vengono imposte sulle facce di tale dominio. tuttavia per il problema di interesse è necessario che l'efflusso avvenga su una regione circolare di dimensione definita, ovvero la trachea. Tale imposizione non può avvenire attraverso una condizione di Dirichlet per problemi di stabilità del codice ma si definisce una condizione di Neumann e si utilizza una "sponge" per forzare il flusso ai valori desiderati. L'obiettivo del primo test era quindi verificare la capacità del codice di imporre una condizione fisica ovvero un profilo desiderato (parabolico nell'esempio) nella regione della trachea e velocità nulla sul resto della faccia. Per quanto riguarda le altre facce la condizione imposta è di Dirichlet e in particolare, come rappresentato in figura A.1, velocità entrante diretta verso un pozzo collocato all'interno del dominio.
La condizione imposta sulla pressione è del tipo di Dirichlet $p = 0$ su tutte le facce del dominio. Il metodo della "sponge" consiste nel definire una regione interna al dominio che ha il compito di modificare le equazioni in modo che sul contorno desiderato si raggiunga la condizione chiesta. Per controllare questa imposizione si utilizzano tre parametri:

- lo spessore della sponge
- lo spessore della regione di transizione della sponge

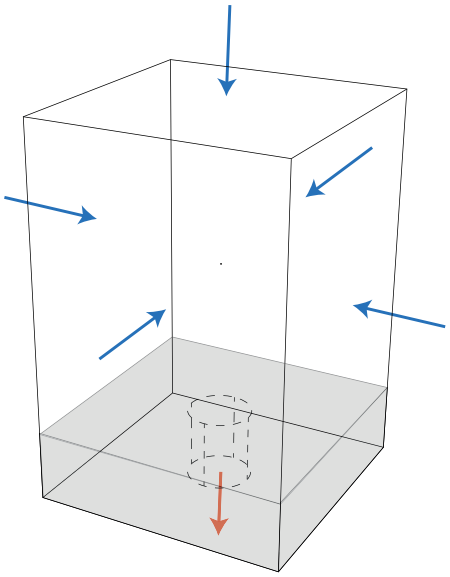


Figura A.1: *Set-up primo test*

- l'intensità dell'imposizione

i primi due parametri sono di carattere geometrico e rappresentano rispettivamente lo spessore totale della regione di sponge e lo spessore della regione di transizione della sponge. Quest'ultima è una regione della sponge nella quale il fluido arriva all'imposizione della condizione con un intensità che parte da 0 e arriva linearmente al valore definito. L'ultimo parametro, ovvero l'intensità dell'imposizione della sponge, varia da zero a uno e determina con quanta forza è necessario imporre la condizione chiesta.

A.1.2 Setup del test

Il test è stato eseguito nel dominio:

$$\begin{aligned} L_x &= 40 \\ L_y &= 40 \\ L_z &= 60 \end{aligned}$$

utilizzando un solo processore con 40 punti lungo x e y e 60 lungo z. il passo temprale utilizzato è $\Delta t = 0.005$ e il tempo di simulazione complessivo è $T = 250$. Inoltre il numero di Reynolds della simulazione (da intendersi come il reciproco della viscosità del fluido) è tenuto basso in queste prove preliminare per non allungare inutilmente i tempi di calcolo. In particolare in questa simulazione si è posto $Re = 60$ dove ricordiamo che nel codice utilizzato $Re = \frac{1}{\nu}$. La velocità che converge al pozzo posizionato in $\boldsymbol{x}_f$ è stato imposto utilizzando la funzione:

$$\boldsymbol{u} = U_0 \frac{\boldsymbol{x}_f - \boldsymbol{x}}{\|\boldsymbol{x}_f - \boldsymbol{x}\|^3} \tag{A.1}$$

con:

$$\begin{aligned} U_0 &= 20 \\ \boldsymbol{x}_f &= \begin{cases} x_f = 20 \\ y_f = 20 \\ z_f = 20 \end{cases} \end{aligned}$$

Mentre utilizzando la sponge sulla faccia L_z si è imposto efflusso solo su un disco di raggio r_0 con un profilo di velocità parabolico:

$$\begin{cases} w = w_0 \left(1 - \dfrac{r^2}{r_0^2}\right) & \text{per } r \leq r_0 \\ w = 0 & \text{altrove} \end{cases} \tag{A.2}$$

con:

$$\begin{aligned} w_0 &= 4.75901 \\ r_0 &= 5 \\ r &= (x - x_c)^2 + (y - y_c)^2 \\ x_c &= 20 \\ y_c &= 20 \end{aligned}$$

dove la velocità massima w_0 è stato calcolata in modo da bilanciare il flusso di massa in ingresso e uscita.
Nel codice tali impostazioni sono state ottenute modificando il file *solution.f90*, in particolare si riportano i segmenti di codice significativi. L'imposizione della sponge avviene grazie alla routine *computesponge*:

```
  SUBROUTINE  compute_sponge(sponge,mesh,t,vn)

    USE types
    USE mesh_mod
    USE toolbox
    USE data

    IMPLICIT NONE

    TYPE(vector_bloc),                INTENT(INOUT) :: sponge
    TYPE(mesh_types) ,                INTENT(IN)    :: mesh
    REAL(KIND=8)     ,                INTENT(IN)    :: t
    TYPE(vector_bloc), DIMENSION(:),  INTENT(IN)    :: vn
    INTEGER                          :: i, j, k, n, nn, dim, coords(3),
   dir, dir2, dir3
    REAL(KIND=8)                     :: x, y, z, nu

! Declare parameters ------------------------------
    REAL(KIND=8) :: sp_str , sp_fun
    REAL(KIND=8) :: domain_l , sp_wid , sp_tra
    REAL(KIND=8) :: x_sp_1 , x_sp_2
    REAL(KIND=8) :: alpha
    REAL(KIND=8) :: radius, r0max, U0max
! ------------------------------------------------
!alpha=1.0            |      ___|------------|
!                     |    --   |            |
!                     |   -     |            |
!alpha=0.0  __________|---      |            |
!                     ^         ^            ^
!                  x_sp_1    x_sp_2       domain_l
! ------------------------------------------------

! - Set sponge parameters -----------
    sp_str   = 1.0d0
    domain_l = 60.0d0
    sp_tra   = 5.0d0 ! Da modificare  (era 4)
```

```
    sp_wid    = 15.0d0 ! Da modificare (era 12)
    x_sp_1    = domain_l - sp_wid
    x_sp_2    =   x_sp_1 + sp_tra
    r0max     = 8.d0
    IF (t < tstart) THEN
      U0max     = 1.d0 * ImpulsiveStartSmooth(t)
    ELSE
      U0max     = 1.d0
    END IF
! ----------------------------------

    dim = mesh%dim
    dir = sponge%direction
    dir2 = MOD(dir,dim)+1
    nu = param%nu

!-- dim = 2 ------------------------------------------------------------
    IF (dim==2) THEN
       IF (dir==1) THEN
          DO i = 1, mesh%nint(dir2)+1
             coords(2) = (i-1)*mesh%weights(dir,2)
             DO j = 1, mesh%nint(dir)+1
                n = coords(2)+j
                x = mesh%coords(n,1)
                y = mesh%coords(n,2)+0.5d0*mesh%h(dir2,1)%a(i+1)
                sponge%f1(n) = 0d0
             END DO
          END DO
       ELSE IF (dir==2) THEN
          DO i = 1, mesh%nint(dir2)+1
             coords(1) = (i-1)*mesh%weights(dir,2)
             DO j = 1, mesh%nint(dir)+1
                n = coords(1)+j
                CALL OneDtoOneD(mesh, n, dir, nn, 1)
                x = mesh%coords(nn,1)+0.5d0*mesh%h(dir2,1)%a(i+1)
                y = mesh%coords(nn,2)
                sponge%f1(n) = 0d0
             END DO
          END DO
       END IF

    ELSE
!-- dim = 3 ------------------------------------------------------------
       dir3 = MOD(dir2,dim)+1
       IF (dir==1) THEN
          DO k = 1, mesh%nint(dir3)+1
             coords(3) = (k-1)*mesh%weights(dir,3)
             DO i = 1, mesh%nint(dir2)+1
                coords(2) = (i-1)*mesh%weights(dir,2)
                DO j = 1, mesh%nint(dir)+1
                   n = coords(3)+coords(2)+j
                   x = mesh%coords(n,1)
                   y = mesh%coords(n,2)+0.5d0*mesh%h(dir2,1)%a(i+1)
                   z = mesh%coords(n,3)+0.5d0*mesh%h(dir3,1)%a(k+1)
                   ! sponge -------------------------
                   IF      ( z .LE. x_sp_1 ) THEN
                     sp_fun = 0.0d0
                   ELSEIF ( ( z .GT. x_sp_1 ) .AND. &
                            ( z .LT. x_sp_2 ) ) THEN
```

```
95                      alpha  = ( z - x_sp_1 ) / sp_tra
96                      sp_fun = 1.0d0 / ( 1.0d0 + &
97                                         exp( 1.0d0/(alpha-1.0d0)+1.0d0/
    alpha) )
98                    ELSE
99                      sp_fun = 1.0d0
100                   ENDIF
101                   ! --------------------------------
102                   sponge%f1(n) = sp_str * sp_fun * &
103                          (  - vn(1)%f1(n) ) ! In the sponge the
    azimuthal velocity is set to 0 to eliminate the problem at the exit
104 !                         ( velocity_u(x,y,z,t) - vn(1)%f1(n) )
105                 ! --------------------------------
106               END DO
107             END DO
108           END DO
109         ELSE IF (dir==2) THEN
110           DO k = 1, mesh%nint(dir3)+1
111             coords(3) = (k-1)*mesh%weights(dir,3)
112             DO i = 1, mesh%nint(dir2)+1
113               coords(2) = (i-1)*mesh%weights(dir,2)
114               DO j = 1, mesh%nint(dir)+1
115                 n = coords(3)+coords(2)+j
116                 CALL OneDtoOneD(mesh, n, dir, nn, 1)
117                 x = mesh%coords(nn,1)+0.5d0*mesh%h(dir3,1)%a(k+1)
118                 y = mesh%coords(nn,2)
119                 z = mesh%coords(nn,3)+0.5d0*mesh%h(dir2,1)%a(i+1)
120                 ! sponge -------------------------
121                 IF      ( z .LE. x_sp_1 ) THEN
122                   sp_fun = 0.0d0
123                 ELSEIF ( ( z .GT. x_sp_1 ) .AND. &
124                          ( z .LT. x_sp_2 ) ) THEN
125                   alpha  = ( z - x_sp_1 ) / sp_tra
126                   sp_fun = 1.0d0 / ( 1.0d0 + &
127                      exp ( 1.0d0/(alpha-1.0d0)+1.0d0/alpha) )
128                 ELSE
129                   sp_fun = 1.0d0
130                 ENDIF
131                 ! --------------------------------
132                 sponge%f1(n) = sp_str * sp_fun * &
133                        ( - vn(2)%f1(n) )! In the sponge the azimuthal
    velocity is set to 0 to eliminate the problem at the exit
134 !                         ( velocity_v(x,y,z,t) - vn(2)%f1(n) )
135                 ! --------------------------------
136               END DO
137             END DO
138           END DO
139         ELSE
140           DO k = 1, mesh%nint(dir3)+1
141             coords(3) = (k-1)*mesh%weights(dir,3)
142             DO i = 1, mesh%nint(dir2)+1
143               coords(2) = (i-1)*mesh%weights(dir,2)
144               DO j = 1, mesh%nint(dir)+1
145                 n = coords(3)+coords(2)+j
146                 CALL OneDtoOneD(mesh, n, dir, nn, 1)
147                 x = mesh%coords(nn,1)+0.5d0*mesh%h(dir2,1)%a(i+1)
148                 y = mesh%coords(nn,2)+0.5d0*mesh%h(dir3,1)%a(k+1)
149                 z = mesh%coords(nn,3)
150                 ! sponge -------------------------
```

```
                    IF     ( z .LE. x_sp_1 ) THEN
                      sp_fun = 0.0d0
                    ELSEIF ( ( z .GT. x_sp_1 ) .AND. &
                             ( z .LT. x_sp_2 ) ) THEN
                      alpha  = ( z - x_sp_1 ) / sp_tra
                      sp_fun = 1.0d0 / ( 1.0d0 + &
                         exp ( 1.0d0/(alpha-1.0d0)+1.0d0/alpha) )
                    ELSE
                      sp_fun = 1.0d0
                    ENDIF
                    ! --------------------------------
                    sponge%f1(n) = sp_str * sp_fun * &
                    !   (U0max - vn(3)%f1(n))
                     ( velocity_w( (/x,y,z/),t ) - vn(3)%f1(n) ) ! Da
   definire funzione velocity_w
                    ! --------------------------------
                END DO
             END DO
          END DO
       END IF

    END IF

  END SUBROUTINE compute_sponge
```

per la quale è necessario definire nel nostro caso la funzione della velocità w:

```
FUNCTION Velocity_w( x,t )
    IMPLICIT NONE
    REAL(KIND=8), DIMENSION(3),  INTENT(IN)  :: x
    REAL(KIND=8),                INTENT(IN)  :: t
    REAL(KIND=8)                             :: Velocity_w
    REAL(KIND=8)                             :: w0, r, a, xc, yc
    xc = 20.d0
    yc = 20.d0
    w0 = 4.75901 !velocitá massima
    r  = 5.0d0 !raggio foro d'uscita
    a  = -w0 / r**2
    IF ((x(1) - xc)**2 + (x(2) - yc)**2 <= r**2) THEN
      Velocity_w = a * (x(1) - xc)**2 + a * (x(2) - yc)**2 + w0
    ELSE
      Velocity_w = 0.0d0
    END IF
  END FUNCTION Velocity_w
```

Per le condizioni al contorno sulle altre facce invece è stato necessario modificare le funzioni *Dirichlet_u*, *Dirichlet_v* e *Dirichlet_w*:

```
FUNCTION Dirichlet_u(x,t)
    IMPLICIT NONE
    REAL(KIND=8),      INTENT(IN)  :: x(3), t
    REAL(KIND=8)                   :: Dirichlet_u
    REAL(KIND=8)                   :: xf, yf, zf, U0

    != Prescribe boundary conditions
    xf = 20.d0
    yf = 20.d0
    zf = 20.d0
```

```
11      U0 = 20.d0
12      IF (x(2) >= 39.9d0) THEN
13        Dirichlet_u = 0.0d0
14      ELSE
15        Dirichlet_u = U0 * (xf - x(1)) /  ((xf - x(1))**2 + (yf - x(2))**2
      + (zf - x(3))**2)**(3.0d0/2.0d0)
16      END IF
17
18    END FUNCTION Dirichlet_u
19
20    !#########################################################################!
21
22    FUNCTION Dirichlet_v(x,t)
23      IMPLICIT NONE
24      REAL(KIND=8),       INTENT(IN)  :: x(3), t
25      REAL(KIND=8)                    :: Dirichlet_v
26      REAL(KIND=8)                    :: xf, yf, zf, U0
27
28      != Prescribe boundary conditions
29      xf = 20.d0
30      yf = 20.d0
31      zf = 20.d0
32      U0 = 20.d0
33
34      IF (x(2) >= 39.9d0) THEN
35        Dirichlet_v = 0.0d0
36      ELSE
37        Dirichlet_v = U0 * (yf - x(2)) /  ((xf - x(1))**2 + (yf - x(2))**2
      + (zf - x(3))**2)**(3.0d0/2.0d0)
38      END IF
39
40    END FUNCTION Dirichlet_v
41
42    !#########################################################################!
43
44    FUNCTION Dirichlet_w(x,t)
45      IMPLICIT NONE
46      REAL(KIND=8),       INTENT(IN)  :: x(3), t
47      REAL(KIND=8)                    :: Dirichlet_w
48      REAL(KIND=8)                    :: xf, yf, zf, U0
49
50      != Prescribe boundary conditions
51      xf = 20.d0
52      yf = 20.d0
53      zf = 20.d0
54      U0 = 20.d0
55
56      IF (x(2) >= 39.9d0) THEN
57        Dirichlet_w = 0.0d0
58      ELSE
59        Dirichlet_w = U0 * (zf - x(3)) /  ((xf - x(1))**2 + (yf - x(2))**2
      + (zf - x(3))**2)**(3.0d0/2.0d0)
60      END IF
61
62    END FUNCTION Dirichlet_w
```

A.1.3 Risultati del test

Il test ha avuto successo, la velocità ottenuta all'efflusso è risultata inferiore a 0.15 su tutta la faccia d'efflusso tranne nel disco imposto dove ha raggiunto i valori richiesti il cui massimo è appunto 4.75901. In termini di portata solo il 2% esce dal dominio attraverso la faccia L_z ma fuori dalla trachea, il risultato è mostrato in figura A.2

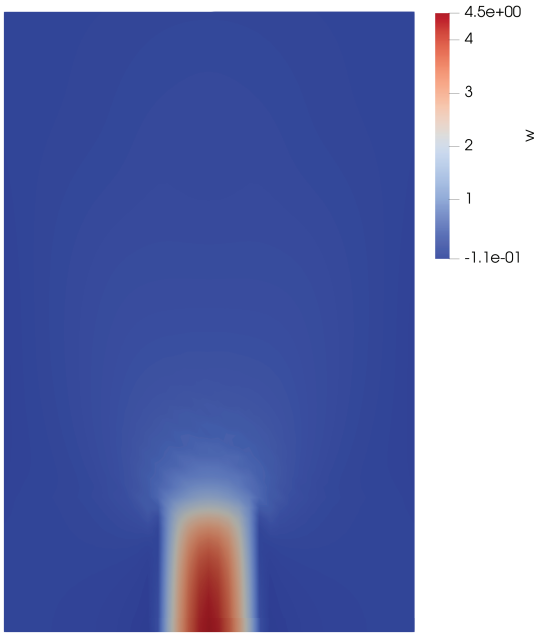


Figura A.2: *Risultato primo test*

A.2 Test geometria

A.2.1 Obiettivo del test

Una volta verificato la capacità del software di imporre le condizioni necessarie per il problema di interesse si è proseguito con i test per indagare il funzionamento del codice includendo il metodo dei contorni immersi (finora inutilizzato). A questo scopo si è inserita nel dominio la geometria che inizialmente era diversa da quella utilizzata nel caso presentato nel capitolo 5, in particolare le cavità nasali risultano uguali ma in questa fase non tutta la testa era presente. Una raffigurazione del dominio e della geometria inserita è mostrata in figura A.3.

É importante notare che per come è implementato il metodo dei contorni immersi nel codice utilizzato non è possibile intersecare la geometria con le pareti del dominio, ma è necessario lasciare almeno una cella di distanza. Quindi sia le "pareti" piatte della testa che la trachea non toccano i contorni del dominio ma sono staccate di poco più di una cella per garantire il funzionamento dell'algoritmo. L'ipotesi di fondo quindi è che se tale spazio è sufficientemente piccolo il flusso non dovrebbe subire particolari effetti dalla presenza di questi "condotti artificiali".

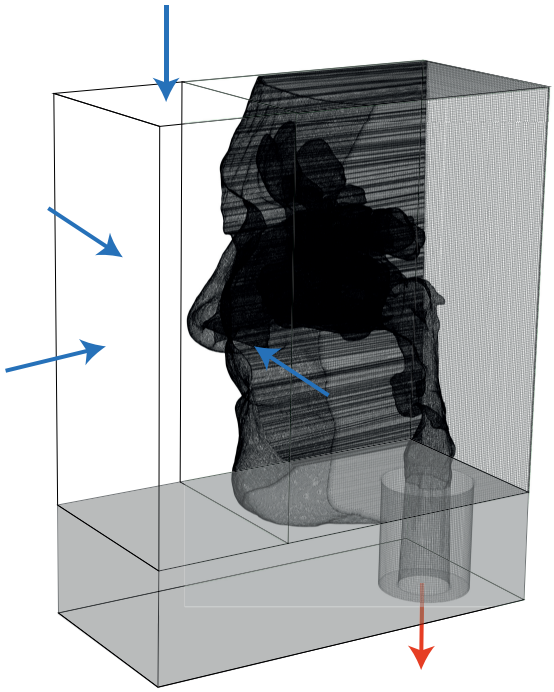


Figura A.3: *Geometria test su geometria reale*

A.2.2 Set-up generale

Il test è stato eseguito nel dominio:

$$\begin{aligned} L_x &= 82 \\ L_y &= 210 \\ L_z &= 215 \end{aligned}$$

utilizzando 2 processori con 27 punti ciascuno in direzione x, 2 processori con 75 punto ciascuno lungo y e 3 processori con 50 punti ciascuno in direzione z. il passo temprale utilizzato è $\Delta t = 10^{-6}$ mentre il tempo di simulazione complessivo varia per problemi di convergenza come verrà specificato più avanti. Il numero di Reynolds della simulazione (da intendersi come il reciproco della viscosità del fluido) è $Re = 0.001136$. Le condizioni al contorno imposte sulla pressione sono ovunque $p = 0$ mentre è stata imposta sulle porzioni delle facce di inflow (indicate con una freccia in figura A.3 una velocità che converge a un pozzo posizionato in $\boldsymbol{x}_f$ utilizzando ancora una volta la funzione:

$$\boldsymbol{u} = U_0 \frac{\boldsymbol{x}_f - \boldsymbol{x}}{\|\boldsymbol{x}_f - \boldsymbol{x}\|^3} \tag{A.3}$$

con:

$$\begin{aligned} U_0 &= 32\,956.6 \\ \boldsymbol{x}_f &= \begin{cases} x_f = 37.4538 \\ y_f = 123.525 \\ z_f = 96.3844 \end{cases} \end{aligned}$$

dove la velocità U_0 è stato calcolata in modo da bilanciare il flusso di massa in ingresso e uscita. Sulla faccia di efflusso, L_z, si è nuovamente utilizzata la sponge per imporre efflusso solo su un

disco di raggio r_0 con un profilo di velocità parabolico:

$$\begin{cases} w = w_0 \left(1 - \dfrac{r^2}{r_0^2}\right) & \text{per } r \leq r_0 \\ w = 0 & \text{altrove} \end{cases} \tag{A.4}$$

con:

$$\begin{aligned} w_0 &= 1101 \\ r_0 &= 10.75 \\ r &= (x - x_c)^2 + (y - y_c)^2 \\ x_c &= 32.56 \\ y_c &= 28.75 \end{aligned}$$

dove la velocità massima w_0 è stata ottenuta dai risultati disponibili e dalla letteratura medica **2EJLJ2013**. I parametri sono stati impostati nel codice come mostrato nella precedente sezione aggiustando dove necessario i valori da imporre.

A.2.3 Risultati del test

Il risultato ottenuto non è stato positivo: il codice ha presentato problemi di convergenza e in particolare si è ottenuta una divergenza nella velocità.

A.2.4 Test successivi e identificazione del problema

Inizialmente si è provato a modificare il passo temprale e ad inserire un coefficiente dipendente dal tempo che rendesse più smooth l'imposizione delle condizioni al contorno arrivando da zero al valore desiderato in un certo numero di passo temprale e non impulsivamente. Tutta via tale prove non hanno dato risultati positivi e quindi, per comprendere meglio la natura del problema, si è deciso di semplificare il test modificando la geometria. In particolare mantenendo fisso il test case si sono sviluppate e testate le due geometrie rappresentate nelle figure A.4 e A.5.

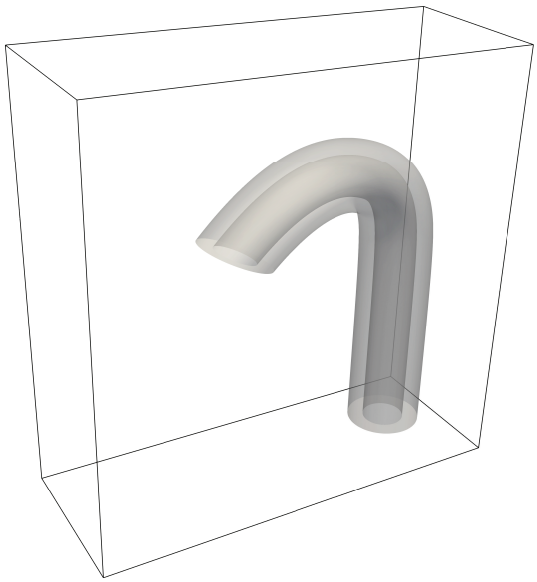


Figura A.4: *Condotto artificiale generato per i test*

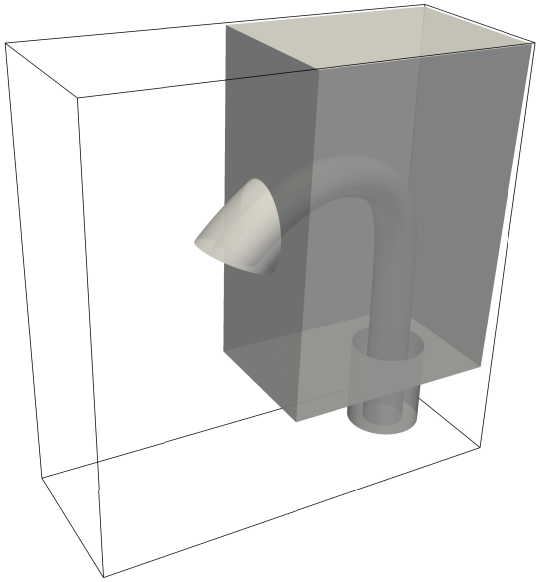


Figura A.5: *Condotto artificiale con scatola generato per i test*

La prima ha in comune con la geometria reale solamente la sezione di efflusso e il fatto che il centro della sezione di inflow corrisponde al punto tra le due narici della geometria reale. Mentre la seconda condivide anche la struttura "a scatola" della geometria reale pur rimanendo geometricamente semplice.
I test effettuati su queste geometrie hanno identificato il problema nei "canali artificiali" tra la geometria immersa e le pareti del dominio, in particolare l'aspetto più critico è la sezione di efflusso infatti in quella zona la depressione causata dalla velocità provoca un risucchio del fluido che si incanala tra la geometria e la trachea con il risultato di un accelerazione che porta rapidamente la velocità a divergere. mentre per evitare il problema sulle superfici della testa è sufficiente ampliare il dominio, per motivi fisici non è possibile allontanare la trachea dalla faccia di efflusso poiché le cavità nasali dovrebbero comunicare con il dominio solo attraverso le narici.
Per risolvere il problema si è pensato di modificare l'implementazione del metodo dei contorni immersi in modo da rendere possibile l'intersezione tra la geometria immersa e le pareti del dominio tuttavia dati i tempi e gli sforzi stimati per tale modifica il gruppo ha deciso di non intraprendere tale strada poiché, come già sottolineato varie volte non c'è interesse pratico nel funzionamento del codice utilizzato in questo lavoro.

A.3 Test forzante

Per evitare il problema riscontrato si è allora deciso di cambiare l'approccio di forzamento, ovvero rinchiudere completamente la geometria nel dominio senza permettere scambio di massa con l'esterno e forzare il flusso attraverso una forza di volume collocata in un apposito prolungamento della gola. Questo approccio necessita un dominio più grande per evitare che si creino flussi secondari tra gola e naso e in questo modo si è raggiunto il test case finale presentato nel capitolo 5.
Questo test pur convergendo numericamente, ha rilevato un ulteriore problema ovvero che l'implementazione del metodo dei contorni immersi non era in grado di imporre correttamente le condizioni al contorno sulla geometria inserita. Tale problema è risultato presente anche in tutti i test precedenti nei quali non era stato notato poiché l'interesse era focalizzato sul problema della divergenza. Come già discusso nel capitolo 5 poiché questo baco non degrada drammatica-

mente a qualità dei risultati ottenuti in questo lavoro e la sua soluzione non è di alcun interesse pratico per il gruppo di ricerca si è deciso, dopo alcuni tentativi preliminari, di non affrontare il debug e la ricerca di una soluzione.

APPENDICE B

Routine di output in pvts

In questa appendice viene presentata la routine scritta in fortran90 per l'esportazione dei risultati del codice disponibile in formato pvts. Si è deciso di sviluppare tale funzione per far fronte al grande dispendio di tempo nell'importazione dei risultati nel software di visualizzazione. Infatti data la quantità di test condotti sul codice, anche utilizzando un numero ristretto di processori, il tempo di organizzazione dei file vtk di output e le operazioni di filtraggio per ottenere i campi di interesse in ParaView risultava dominante nell'analisi dei risultati delle fasi preliminari.

B.1 Il formato vtk

Il formato vtk (Visualization Tool-Kit) **BVTK11** rappresenta uno standard molto diffuso per l'esportazione e successiva visualizzazione dei dati, nel corso degli anni esso si è evoluto per far fronte a file di risultati sempre più grandi come dimensione e prodotti tipicamente da macchine multiprocessore che per questioni di efficienza e costi hanno come output diversi file per ogni processore utilizzato. L'evoluzione del formato oggi denominato *legacy vtk* è avvenuta attraverso quelli che vengono definiti XML-based vtk ovvero file di dati organizzati in una struttura specificata con la sintassi dell XML. Questa tipologia di file risulta più comoda per l'esportazione dei risultati di calcoli paralleli poiché consente di specificare la struttura del dominio complessivo direttamente nel file e di non doverla ricostruire a posteriori. I formati disponibili a seconda del tipo di dati utilizzati sono:

- *Legacy vtk (.vtk)*: File per l'esportazione di risultati seriali formattati come vtkLegacy
- *ImageData (.vti)*: File per l'esportazione di risultati seriali formattati come vtkImageData (griglia strutturata).
- *PolyData (.vtp)*: File per l'esportazione di risultati seriali formattati come vtkPolyData (griglia non strutturata).
- *RectilinearGrid (.vtr)*: File per l'esportazione di risultati seriali formattati come vtkRectilinearGrid (griglia strutturata).

- *StructuredGrid (.vts)*: File per l'esportazione di risultati seriali formattati come vtkStructuredGrid (griglia strutturata).
- *UnstructuredGrid (.vtu)*: File per l'esportazione di risultati seriali formattati come vtkUnstructuredGrid (griglia non strutturata).

Per questo lavoro i due tipi di dataset di interesse sono il vtk Legacy che è il formato di esportazione attuale del codice e il vts o più propriamente il pvts che rappresenta l'obiettivo da raggiungere. Tutti i formati XML presentati hanno una controparte parallela (con estensione p"estensione", ad esempio vts → pvts) che consente di specificare la struttura complessiva del dominio di interesse richiamando i singoli file seriali di ogni processore.

B.2 L'attuale routine di esportazione

L'attuale routine di esportazione è basata sul formato vtk legacy versione 3.0 e per ogni processore, per ogni variabile scalare esporta un file vtk di questo tipo:

```
# vtk DataFile Version 3.0
vtk u
ASCII
DATASET STRUCTURED_GRID
DIMENSIONS     110      142      105
POINTS 1640100 float

coordinate dei punti dove si definisce il valore della variabile
x   y   z

POINT_DATA 1640100
SCALARS u float
LOOKUP_TABLE default

valori di u nei punti indicati sopra
```

Per utilizzare questo tipo di file in ParaView per visualizzare i campi di pressione e velocità sul dominio è necessario:

1. includere tutti i file: quattro per ogni processore (u, v, w e p)
2. unire i file di ogni componente utilizzando il filtro di ParaView *GroupAttribute*
3. unire le tre componenti della velocità per ottenere il campo vettoriale utilizzando il filtro *AppendAttribute*

Tale procedura risulta dispendiosa in termini di tempo poiché necessita l'applicazione di un filtro per ogni componente per ogni processore più uno ulteriore per ogni componente per un totale di $4N_p + 3$ che già utilizzando 6 processori sono 27 filtri per la sola visualizzazione dei campi. Inoltre i campi così ottenuti sono legati dai suddetti filtri dando origine a una struttura in ParaView su cui è difficile applicare filtri più complessi per estrarre informazioni di interesse. per questi motivi si è scelto di passare al formato pvts.

B.3 Il formato vts

Il formato scelto è quello che permette di descrivere griglie di tipo strutturato, la sua implementazione parallela avviene nel modo seguente

- Per ogni processore viene prodotto un file vts che contiene sia le coordinate dei punti della griglia che i valori delle variabili scalari, vettoriali o tensoriali di interesse

- Viene generato un unico file con estensione pvts che contiene le informazioni sul dominio globale e si occupa di caricare i dati dai corretti file vts

L'utilizzo in ParaView di questo formato è molto semplice, aprendo il file pvts automaticamente vengono importati tutti i risultati la velocità già sotto forma di campo vettoriale e la pressione come campo scalare. I file vts sono del tipo:

```
<VTKFile type="StructuredGrid" version="1.0" byte_order="LittleEndian">
<StructuredGrid WholeExtent="110 219  0 141  209 313 ">
<Piece Extent="110 219  0 141  209 313 ">
<PointData Vectors="Velocity" Scalars="Pressure">
<DataArray type="Float32" Name="Velocity" NumberOfComponents="3" format="
    ascii">

componeneti della velocita'
u v w

</DataArray>
<DataArray type="Float32" Name="Pressure" format="ascii">

valori di pressione
p

</DataArray>
</PointData>
<CellData>
</CellData>
<Points>
<DataArray type="Float32" Name="Points" NumberOfComponents="3" format="
    ascii">

coordinate dei punti di griglia
x y z

</DataArray>
</Points>
</Piece>
</StructuredGrid>
</VTKFile>
```

La prima riga del file specifica il tipo di file che si sta utilizzando, la seconda determina l'estensione del dominio sul quale è valido il file (nel caso di ciascun processore il proprio sottodominio), la terza riga specifica, se necessarie, una o più sotto-regioni del dominio. Le estensioni geometriche sono fornite come indici i, j e k nelle tre direzioni come di consueto per griglie strutturate. Nella quarta riga sono definite le variabili che verranno specificate nelle righe successive. Si ha poi la definizione delle coordinate nelle quali sono definite le variabili sopra indicate. Il file pvts è del tipo:

```
<VTKFile type="PStructuredGrid" version="0.1" byte_order="LittleEndian">
<PStructuredGrid WholeExtent="0 219   0 281  0 314  " GhostLevel="#">
<PPointData Vectors="Velocity" Scalars="Pressure">
<PDataArray type="Float32" Name="Velocity" NumberOfComponents="3" format
    ="ascii"/>
<PDataArray type="Float32" Name="Pressure" />
</PPointData>
<PCellData></PCellData>
<PPoints>
```

```
<PDataArray  type="Float32" Name="coordinate" NumberOfComponents="3"
   format="ascii" />
</PPoints>
<Piece Extent="0 110  0 141  0 105 " Source="a_00.vts">
</Piece>
<Piece Extent="0 110  0 141  105 209 " Source="a_01.vts">
</Piece>
<Piece Extent="0 110  0 141  209 313 " Source="a_02.vts">
</Piece>
<Piece Extent="0 110  141 281  0 105 " Source="a_03.vts">
</Piece>
<Piece Extent="0 110  141 281  105 209 " Source="a_04.vts">
</Piece>
<Piece Extent="0 110  141 281  209 313 " Source="a_05.vts">
</Piece>
<Piece Extent="110 219  0 141  0 105 " Source="a_06.vts">
</Piece>
<Piece Extent="110 219  0 141  105 209 " Source="a_07.vts">
</Piece>
<Piece Extent="110 219  0 141  209 313 " Source="a_08.vts">
</Piece>
<Piece Extent="110 219  141 281  0 105 " Source="a_09.vts">
</Piece>
<Piece Extent="110 219  141 281  105 209 " Source="a_10.vts">
</Piece>
<Piece Extent="110 219  141 281  209 313 " Source="a_11.vts">
</Piece>
</PStructuredGrid>
</VTKFile>
```

In questo tipo di file la prima riga specifica il formato che si intende utilizzare, la seconda l'estensione dell'intero dominio, la terza indica le variabili di interesse e il loro tipo. Le righe successive definiscono il modo in cui sono fornite le coordinate dei punti della griglia e i file vts da cui prendere le informazioni relative a ciascun sotto-dominio.

B.4 La nuova routine di esportazione

Di seguito si presenta il sorgente della routine implementata nel solutore per produrre i file necessari all'esportazione.

```
1  SUBROUTINE create_vts_file(fieldu, fieldv, fieldw, fieldp, mesh,
       file_name, t, one_d_rank)
2    USE MPI
3    USE types
4    USE mesh_mod
5    USE toolbox
6    USE util
7    IMPLICIT NONE
8    TYPE(vector_bloc), INTENT(INOUT) :: fieldu, fieldv, fieldw, fieldp
9    TYPE(mesh_types),  INTENT(IN)  :: mesh
10   CHARACTER(*),      INTENT(IN) :: file_name
11   INTEGER, DIMENSION(:), INTENT(IN)     :: one_d_rank
12   INTEGER                 :: i, j, k, d, dim, &
13                     dir1, dir2, dir3, dir1u, dir2u,&
14                     dir3u, dir1v,dir2v, dir3v,&
15                     dir1w, dir2w, dir3w, dir1p, dir2p, dir3p,&
```

```
16                    ns1, ns2, ns3, ns1u, ns2u, ns3u, ns1v, ns2v, ns3v, &
17                    ns1w, ns2w, ns3w, ns1p, ns2p, ns3p, &
18                    X1, X2, X3, X1u, X2u, X3u, X1v, X2v, X3v, X1w, X2w,
    X3w, X1p, X2p, X3p, &
19                    xs, xe, ys, ye, zs, ze, & !indice inizio e fine
    ciascuno subset del core
20                    xm, ym, zm, & ! coordinate di fine del dominio
    globale(l'inizio e' sempre 0 0 0)
21                    xp, yp, zp, & !variabili ausiliarie
22                    nx, ny, nz, code, rank, l, lblank, start
23  REAL(KIND=8)            :: id !mega trick
24  INTEGER               :: end_line(3)
25  REAL(KIND=8)            :: x, y, z, t, xoffset, yoffset, zoffset
26  CHARACTER(len=4)          :: tit
27  INTEGER               :: unit_file, nb_procs
28  dim = mesh%dim
29
30  CALL MPI_Comm_rank(MPI_COMM_WORLD, rank, code)
31  CALL MPI_Comm_size(MPI_COMM_WORLD, nb_procs, code)
32
33  WRITE(tit,'(I4)') rank
34  lblank = eval_blank(4,tit)
35  DO l = 1, lblank - 1
36    tit(l:l) = '0'
37  END DO
38  start = 4 - nb_digit() + 1
39  !CALCOLO ESTREMI SUBSET DEL CORE:
40
41  DO d = 1, dim
42    IF (mesh%BCtype(2,d)/=0) THEN
43      end_line(d) = mesh%nint(d) + 1
44    ELSE
45      end_line(d) = mesh%nint(d) + 2
46    END IF
47  END DO
48
49  !inizio calcolo estremi
50
51  id = rank + 1
52  ny = mesh%Ndom(2)
53  nz = mesh%Ndom(3) !queste due variabili poi verranno riscritte, é
     normale
54
55  xp = CEILING(id / (ny * nz))
56  yp = CEILING(  (id - ((xp - 1)*ny*nz)) / nz  )
57  zp = id - (xp - 1)*ny*nz - (yp - 1)*nz
58
59  nx =  end_line(1)-1
60  ny =  end_line(2)-1
61  nz =  end_line(3)-1
62
63  IF (xp==1) nx = nx - 1
64  IF (yp==1) ny = ny - 1
65  IF (zp==1) nz = nz - 1
66
67  xs = (xp - 1) * nx + 1
68  xe =  xp * nx + 1
69  IF (xp==1) xs = 0
70
```

```
71  ys = (yp -1) * ny + 1
72  ye =  yp * ny + 1
73  IF (yp==1) ys = 0
74
75  zs = (zp -1) * nz + 1
76  ze =  zp * nz + 1
77  IF (zp==1) zs = 0
78
79  nx =  end_line(1) !mesh%nint(1) + 2
80  ny =  end_line(2) !mesh%nint(2) + 2
81  nz =  end_line(3) !mesh%nint(3) + 2
82
83  !FINE CALCOLO ESTREMI
84  unit_file=rank+10
85  OPEN (UNIT=unit_file, FILE=file_name//'_'//tit(start:)//'.vts', FORM =
      'formatted', STATUS = 'unknown')
86  WRITE(unit_file,'(A)') '<VTKFile type="StructuredGrid" version="1.0"
      byte_order="LittleEndian">'
87  WRITE(unit_file,*) '<StructuredGrid WholeExtent="',xs,xe,ys,ye,zs,ze,'
      ">'
88  WRITE(unit_file,*) '<Piece Extent="',xs,xe,ys,ye,zs,ze,'">'
89  WRITE(unit_file,'(A)')'<PointData Vectors="Velocity" Scalars="Pressure
      ">'
90  !velocita'
91  WRITE(unit_file,'(A)') '<DataArray type="Float32" Name="Velocity"
      NumberOfComponents="3" format="ascii">'
92
93  dir1u = fieldu%direction
94  dir2u = MOD(dir1u,dim)+1
95  dir3u = MOD(dir2u,dim)+1
96
97  dir1v = fieldv%direction
98  dir2v = MOD(dir1v,dim)+1
99  dir3v = MOD(dir2v,dim)+1
100
101 dir1w = fieldw%direction
102 dir2w = MOD(dir1w,dim)+1
103 dir3w = MOD(dir2w,dim)+1
104
105 dir1p = fieldp%direction
106 dir2p = MOD(dir1p,dim)+1
107 dir3p = MOD(dir2p,dim)+1
108
109 xoffset = 0.5d0
110 yoffset = 0.5d0
111 zoffset = 0.5d0
112
113 X1u = MOD(1-dir1u+dim,dim) + 1
114 X2u = MOD(2-dir1u+dim,dim) + 1
115 X3u = MOD(3-dir1u+dim,dim) + 1
116
117 X1v = MOD(1-dir1v+dim,dim) + 1
118 X2v = MOD(2-dir1v+dim,dim) + 1
119 X3v = MOD(3-dir1v+dim,dim) + 1
120
121 X1w = MOD(1-dir1w+dim,dim) + 1
122 X2w = MOD(2-dir1w+dim,dim) + 1
123 X3w = MOD(3-dir1w+dim,dim) + 1
124
```

```
125   X1p = MOD(1-dir1p+dim,dim) + 1
126   X2p = MOD(2-dir1p+dim,dim) + 1
127   X3p = MOD(3-dir1p+dim,dim) + 1
128
129   !scrivo il campo(valori) velocita'
130
131   DO k = 1, end_line(3) !mesh%nint(3) + 2
132     ns3u = (k-1)*mesh%weights(dir1u,X3u) + 1
133     ns3v = (k-1)*mesh%weights(dir1v,X3v) + 1
134     ns3w = (k-1)*mesh%weights(dir1w,X3w) + 1
135     DO j = 1, end_line(2) !mesh%nint(2) + 2
136       ns2u = ns3u + (j-1)*mesh%weights(dir1u,X2u)
137       ns2v = ns3v + (j-1)*mesh%weights(dir1v,X2v)
138       ns2w = ns3w + (j-1)*mesh%weights(dir1w,X2w)
139       DO i = 1, end_line(1) ! mesh%nint(1) + 2
140         ns1u = ns2u + (i-1)*mesh%weights(dir1u,X1u)
141         ns1v = ns2v + (i-1)*mesh%weights(dir1v,X1v)
142         ns1w = ns2w + (i-1)*mesh%weights(dir1w,X1w)
143         WRITE(unit_file,'(3(e13.7,2x),A))') fieldu%f1(ns1u), fieldv%f1(
    ns1v), fieldw%f1(ns1w)
144       END DO
145     END DO
146   END DO
147
148   WRITE(unit_file,'(A)') '</DataArray>'
149
150   !pressione
151   WRITE(unit_file,'(A)') '<DataArray type="Float32" Name="Pressure"
      format="ascii">'
152
153   DO k = 1, end_line(3) !mesh%nint(3) + 2
154     ns3p = (k-1)*mesh%weights(dir1p,X3p) + 1
155     DO j = 1, end_line(2) !mesh%nint(2) + 2
156       ns2p = ns3p + (j-1)*mesh%weights(dir1p,X2p)
157       DO i = 1, end_line(1) ! mesh%nint(1) + 2
158         ns1p = ns2p + (i-1)*mesh%weights(dir1p,X1p)
159         WRITE(unit_file,'(e13.7)') fieldp%f1(ns1p)
160       END DO
161     END DO
162   END DO
163   WRITE(unit_file,'(A)') '</DataArray>'
164   WRITE(unit_file,'(A)') '</PointData>'
165   WRITE(unit_file,'(A)') '<CellData>'
166   WRITE(unit_file,'(A)') '</CellData>'
167
168   !scrivo le coordinate dei punti della mesh
169   WRITE(unit_file,'(A)') '<Points>'
170   WRITE(unit_file,'(A)') '<DataArray type="Float32" Name="Points"
      NumberOfComponents="3" format="ascii">'
171
172   X1 = MOD(1-1+dim,dim) + 1
173   X2 = MOD(2-1+dim,dim) + 1
174   X3 = MOD(3-1+dim,dim) + 1
175   DO k = 1, end_line(3) !mesh%nint(3) + 2
176     ns3 = (k-1)*mesh%weights(1,X3) + 1
177     DO j = 1, end_line(2) !mesh%nint(2) + 2
178       ns2 = ns3 + (j-1)*mesh%weights(1,X2)
179       DO i = 1, end_line(1) !mesh%nint(1) + 2
180         ns1 = ns2 + (i-1)*mesh%weights(1,X1)
```

```
181         x = mesh%coords(ns1,1)+xoffset*mesh%h(1,1)%a(i+1)
182         y = mesh%coords(ns1,2)+yoffset*mesh%h(2,1)%a(j+1) !gli offset
    servono perche' u,v,w sono definiti sulle facce della cella non al
    centro come p
183         z = mesh%coords(ns1,3)+zoffset*mesh%h(3,1)%a(k+1) !ma io
    definisco tutto al centro se no sbatti
184         WRITE(unit_file,'(3(e13.7,2x),A)') x, y, z
185       END DO
186     END DO
187   END DO
188
189   WRITE(unit_file,'(A)') '</DataArray>'
190   WRITE(unit_file,'(A)') '</Points>'
191   WRITE(unit_file,'(A)') '</Piece>'
192   WRITE(unit_file,'(A)') '</StructuredGrid>'
193   WRITE(unit_file,'(A)') '</VTKFile>'
194
195   CLOSE(unit_file)
196
197
198
199   !creo il pvts
200   IF (rank==0) THEN
201     OPEN (UNIT=unit_file, FILE=file_name//'.pvts', FORM = 'formatted',
    STATUS = 'unknown')
202
203     !calcolare whole extent
204     xm = mesh%Ndom(1)*(mesh%nint(1) + 1) -1
205     ym = mesh%Ndom(2)*(mesh%nint(2) + 1) -1
206     zm = mesh%Ndom(3)*(mesh%nint(3) + 1) -1
207     !fine clcolo WholeExtent
208     WRITE(unit_file,'(A)') '<VTKFile type="PStructuredGrid" version="0.1"
     byte_order="LittleEndian">'
209     WRITE(unit_file,*) '<PStructuredGrid WholeExtent="0 ',xm,' 0 ',ym,' 0
     ',zm,' " GhostLevel="#">'
210     WRITE(unit_file,'(A)') '<PPointData Vectors="Velocity" Scalars="
    Pressure">'
211     WRITE(unit_file,'(A)') '<PDataArray type="Float32" Name="Velocity"
    NumberOfComponents="3" format="ascii"/>'
212     WRITE(unit_file,'(A)') '<PDataArray type="Float32" Name="Pressure" />
    '
213     WRITE(unit_file,'(A)') '</PPointData>'
214     WRITE(unit_file,'(A)') '<PCellData></PCellData>'
215     WRITE(unit_file,'(A)') '<PPoints>'
216     WRITE(unit_file,'(A)') '<PDataArray type="Float32" Name="coordinate"
    NumberOfComponents="3" format="ascii" />'
217     WRITE(unit_file,'(A)') '</PPoints>'
218     !ciclo inclusione file
219     DO j = 0, nb_procs-1 ! j indica di quale processore si parla-->
    assieme al numero di processori e punti dovrei sapere i limiti-->
    pExtent
220       WRITE(tit,'(I4)') j
221       lblank = eval_blank(4,tit)
222       DO l = 1, lblank - 1
223         tit(l:l) = '0'
224       END DO
225       start = 4 - nb_digit() + 1
226
227       !calcolo ESTREMI
```

```
228       id = j + 1
229       ny = mesh%Ndom(2)
230       nz = mesh%Ndom(3) !queste due variabili poi verranno riscritte , é
      normale
231
232       xp = CEILING(id / (ny * nz))
233       yp = CEILING(  (id - ((xp - 1)*ny*nz)) / nz  )
234       zp = id - (xp - 1)*ny*nz - (yp - 1)*nz
235
236       nx =  end_line(1) -2
237       ny =  end_line(2) -2
238       nz =  end_line(3) -2
239
240       xs = (xp - 1) * nx + 1
241       xe =  xp * nx + 1
242       IF (xp==1) xs = 0
243
244       ys = (yp -1) * ny + 1
245       ye =  yp * ny + 1
246       IF (yp==1) ys = 0
247
248       zs = (zp -1) * nz + 1
249       ze =  zp * nz + 1
250       IF (zp==1) zs = 0
251       !fine calcolo estermi
252       WRITE(unit_file,*) '<Piece Extent="',xs,xe,ys,ye,zs,ze,'" Source="'
      //file_name//'_'//tit(start:)//'.vts">'
253       WRITE(unit_file,'(A)') '</Piece>'
254     END DO
255
256     WRITE(unit_file,'(A)') '</PStructuredGrid>'
257     WRITE(unit_file,'(A)') '</VTKFile>'
258
259     CLOSE(unit_file)
260   END IF
261 END SUBROUTINE create_vts_file
```

Come validazione per la funzione implementata si sono confrontati i risultati con quelli ottenuti utilizzando la routine precedente al variare del numero di processori e per mesh di dimensioni differenti. Tale confronto non ha mostrato differenza tra i risultati ottenuti.