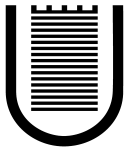
Università degli Studi di Roma "Tor Vergata"
Facoltà di Ingegneria



Dottorato di Ricerca

Ingegneria dell'Energia – Ambiente

Ciclo XX

Analisi di un motore ad alte prestazioni mediante un tool 1D/3D

Candidato

Francesco Del Citto

Coordinatore

Prof. Fabio Gori

Tutor:

Prof. Gino Bella

Indice

Elenco delle figure

Abstract

Numerical simulations of reactive flows are among the most computational demanding applications in the scientific computing world. KIVA–3V, a widely used computer program for CFD, specifically tailored to engine applications, had been deeply modified in order to improve accuracy and stability, while reducing computational time. The original methods included in KIVA to solve equations of fluid dynamics had been fully replaced by new solvers, with the aim of both improving performance and writing a fully parallel code. Almost every feature of original KIVA–3V has been partially or entirely rewritten, a full 1D code has been included and a strategy to link directly 3D zones with zero dimensional models has been developed. Great attention has been given to everything that could lead to a more convenient software, overcoming a lot of limitations of the original version and adding new features to reduce the time needed to prepare input data and use the output, which is a critical aspect in this kind of distributed parallel computation. The result

is a reliable program, noticeably faster than the original KIVA–3V in serial mode and obviously even more in parallel, capable of treating more complex cases and bigger grids, with the desired level of details where required. In the next pages, the new features introduced will be described in details, with special attention to the 3D/1D and 3D/0D coupling strategies. Then, a few examples of the tool's application fields will be presented, showing the benefits of using a multidimensional approach for large and complex problems like high performance internal combustion engines.

Introduzione

Le simulazioni numeriche di problemi di fluidodinamica sono tra le applicazioni più dispendiose, in termini sia di risorse che di tempo di calcolo, tra quelle in uso nella comunità scientifica.

La necessità di aumentare le dimensioni e la complessità dei modelli da un lato e la risoluzione della soluzione dall'altro porta ad un incremento del numero di celle di discretizzazione, quando si usino metodi ai volumi finiti per la risoluzione delle equazioni. Questo comporta, come si può facilmente immaginare, un conseguente aumento del tempo di calcolo necessario.

Il naturale aumento della velocità dei processori nel tempo non è però sufficiente a compensare le maggiori richieste da parte sia degli ambiti di ricerca che del mondo industriale. Da qualche anno è ormai chiaro come un codice di calcolo progettato per rispondere a tali richieste debba necessariamente essere realizzato per un funzionamento in parallelo, permettendo così di compensare le limitate capacità di ogni singolo processore con la possibilità di

suddividere il carico di lavoro tra molte unità e ridurre così significativamente il tempo di attesa tra l'inizio e la fine della simulazione.

La tendenza attuale è quella di affiancare, durante il ciclo di progettazione di un qualsiasi componente meccanico, alle normali tecniche di verifica e analisi anche dei nuovi strumenti di "prototipazione virtuale", in campo fluidodinamico, strutturale e molti altri, che permettono di ridurre il numero di prove fisiche da effettuare prima di trovare una soluzione rispondente alle specifiche richieste. È dunque essenziale che i tempi di attesa per la simulazione del fenomeno fisico in esame siano competitivi con le normali tecniche di prototipazione rapida e test in uso. Ed è altresì importante che l'accuratezza dei risultati e la complessità dei modelli siano sufficientemente elevate da predire correttamente il comportamento del componente.

In particolare, nella progettazione dei motori a combustione interna l'osservazione diretta dei fenomeni fisici che avvengono all'interno del cilindro e in tutti gli altri componenti durante un'intero ciclo di funzionamento è molto complicata e costosa, dovendo far ricorso a prototipi composti da materiali speciali e a strumenti di misura non invasivi, veloci ed accurati.

In questo settore l'analisi CFD dei flussi reagenti all'interno del motore risulta un utilissimo strumento per la comprensione del comportamento di ogni componente prima e per la sua ottimizzazione poi ed è diventato ormai un mezzo imprescindibile per la progettazione di veicoli sempre più efficienti.

Nel corso delle pagine che seguono verranno illustrate le strategie utilizzate per aumentare i campi di utilizzo delle simulazioni di fluidodinamica nel campo della simulazione dei motori a combustione interna.

Da un lato, il codice tridimensionale utilizzato (KIVA) è stato ampliato nelle sue caratteristiche, reso più efficiente con la sostituzione degli algoritmi per risolvere le equazioni ed è stato interamente parallelizzato. Dall'altro il codice 3D è stato accoppiato ad uno 1D efficiente e flessibile, introducendo inoltre una modalità di accoppiamento diretto 3D – 0D.

Il risultato è uno strumento in grado di gestire sistemi complessi, aumentando il dettaglio là dove le geometrie o il problema lo richiedano, potendo contemporaneamente beneficiare della scalabilità e della riduzione dei tempi di calcolo propri del calcolo parallelo.

Di seguito verranno descritti i modelli computazionali, i miglioramenti al codice 3D nel campo della soluzione delle equazioni che descrivono la fisica del problema e le strategie e i risultati della parallelizzazione in tutti i sui aspetti. Successivamente verranno presentate le nuove caratteristiche implementate, inerenti gran parte delle componenti dell'intera procedura di lavoro usuale, nonché l'accoppiamento tra i modelli 3D, 1D e 0D. Verranno infine presentati i risultati ottenuti.

Capitolo 1

Modelli computazionali

1.1 Codice 1D

Il codice monodimensionale utilizzato (1dime) è stato sviluppato presso il DIME[1] dell'Università di Napoli "Federico II" durante molti anni [4–9] ed è in grado di gestire tutte le principali configurazione di motori, grazie ad una struttura modulare estremamente versatile.

La metodologia numerica include un affidabile modello di flusso 1D [24,25] per la caratterizzazione del fenomeno di propagazione delle onde nei tubi, descritto dalle seguenti equazioni:

$$\mathbf{U}_t + [\mathbf{F}(\mathbf{U})]_x = \mathbf{S} \tag{1.1}$$

[1]Dipartimento di Ingegneria Meccanica ed Energetica

dove:

$$\mathbf{U} = \begin{Bmatrix} \rho \\ \rho \mathrm{u} \\ \rho \mathrm{E} \\ \rho \mathrm{x_r} \\ \rho \mathrm{x_f} \end{Bmatrix} \quad \mathbf{F} = \begin{Bmatrix} \rho \mathrm{u} \\ \rho \mathrm{u}^2 + \mathrm{p} \\ \rho \mathrm{uH} \\ \rho \mathrm{ux_r} \\ \rho \mathrm{ux_f} \end{Bmatrix} \tag{1.2}$$

$$\mathbf{S} = - \begin{Bmatrix} \rho \mathrm{u} \dfrac{1}{\Omega} \dfrac{\mathrm{d}\Omega}{\mathrm{dx}} \\ \rho \mathrm{u}^2 \left[\dfrac{1}{\Omega} \dfrac{\mathrm{d}\Omega}{\mathrm{dx}} + \left(2 \dfrac{f}{D} + \dfrac{C_p}{2L} \right) \dfrac{\mathrm{u}}{\|\mathrm{u}\|} \right] \\ \rho \mathrm{uH} \dfrac{1}{\Omega} \dfrac{\mathrm{d}\Omega}{\mathrm{dx}} - 4 \dfrac{q}{D} \\ \rho \mathrm{ux_r} \dfrac{1}{\Omega} \dfrac{\mathrm{d}\Omega}{\mathrm{dx}} \\ \rho \mathrm{ux_f} \dfrac{1}{\Omega} \dfrac{\mathrm{d}\Omega}{\mathrm{dx}} \end{Bmatrix} \tag{1.3}$$

I termini ρ, u, p, $\mathrm{E} = c_v T + \mathrm{u}^2/2$, $\mathrm{H} = c_p T + \mathrm{u}^2/2$ nel sistema di equazioni (1.1) rappresentano, rispettivamente, densità, velocità, pressione, energia totale ed entalpia per unità di massa. Il termine sorgente $\mathbf{S}$ tiene conto della variazione d'area dei condotti $\mathrm{d}\Omega/\mathrm{dx}$, dello scambio di calore q e delle perdite per attrito [4]. Quest'ultimo termine include sia un coefficiente d'attrito f sia un coefficiente di perdita di pressione C_p, determinato come funzione della conicità e della curvatura del condotto [26]. Le ultime due equazioni nel sistema (1.1) rappresentano il trasporto scalare delle specie chimiche, espresse attraverso le frazioni di carburante x_f e di combusti x_r. Queste equazioni

tengono traccia della composizione del gas evolvente in condizioni di flusso diretto o inverso e permettono di stimare la composizione media del gas e il livello di EGR all'interno del cilindro alla chiusura della valvola d'aspirazione. Il sistema di equazioni (1.1) è risolto numericamente in uno schema ai volumi finiti di tipo TVD[2], raggiungendo un'accuratezza del secondo e quarto ordine rispettivamente nello spazio e nel tempo [24, 25].

Il modello monodimensionale include inoltre un trattamento zero – dimensionale multizona dei cilindri. In particolare, la velocità di combustione, nel caso di un motore ad accensione comandata, è calcolata per mezzo di un modello di combustione turbolenta, basato su una descrizione frattale della superficie del fronte di fiamma [5–9, 27]. A seguito di questo approccio, una superficie di forma sferica iniziale del fronte di fiamma, la fiamma laminare A_L, viene increspato dalla presenza di vortici di diversa scala. L'interazione tra il campo di flusso turbolento e la fiamma determina lo sviluppo della fiamma turbolenta, A_T, che si propaga alla velocità laminare di fiamma S_L. Il modello di combustione è stato recentemente esteso all'analisi di motori twin–spark [28].

[2]Total Variation Diminishing

1.2 Codice 3D

Il codice di calcolo 3D utilizzato è una versione profondamente modificata del ben noto KIVA–3V [1, 2]. Di seguito vengono descritte le caratteristiche del codice originale. Per una descrizione delle numerose novità introdotte, sia sotto il profilo dei metodi numerici sia delle funzionalità, si rimanda al capitolo 2.

Il programma KIVA-3V, utilizzato nello svolgimento del presente lavoro, fa parte della famiglia dei modelli multidimensionali; è un codice di calcolo ai volumi finiti che impiega una mesh del tipo multiblocco strutturata. Il KIVA-3V, sviluppato al *Los Alamos National Laboratory* nel 1997, è l'ultimo di una serie di codici fluidodinamici multidimensionali cominciata circa 20 anni fa.

Il primo programma KIVA fu realizzato e commercializzato nel 1985 e successivamente sostituito dallo sviluppo KIVAII nel 1989. Questa versione più recente bene si prestava allo studio di flussi confinati in geometrie cilindriche e ad una varietà di sistemi aperti, ma comunque geometrie semplici; era tutt'altro che efficiente quando applicato a geometrie complesse quali precamere di motori diesel ad iniezione indiretta o profili di condotti di aspirazione o scarico. Questa situazione risultava dal fatto che l'intero dominio doveva essere circondato da una singola matrice di punti (mesh monoblocco) con nu-

mero d'indice fissato nelle tre direzioni[3], con il risultato di ottenere numerosi punti esterni alla regione di interesse che dovevano essere successivamente deattivati.

KIVA3 rimosse questo problema attraverso l'introduzione di una griglia multiblocco, che eliminava interamente la necessità di creare regioni con celle inattive. Inoltre l'uso dell'indirizzamento indiretto per la connettività permise ai vettori contenenti i dati relativi alla griglia di essere ordinati, al fine di minimizzare il numero di nodi sul quale effettuare i cicli ed eliminare i controlli sulle flag in ogni ciclo e per ogni indice. In più, i dati relativi alle condizioni al contorno furono inseriti in una tabella che permise al programma di spazzare sul vettore dei vertici e delle celle individuando direttamente quelli interessati da dette condizioni.

KIVA–3V rappresenta uno sviluppo significativo del codice. Al nucleo di base è stata aggiunta una effettiva modellazione delle valvole di aspirazione e di scarico[4]. Le valvole sono trattate all'interno del codice come degli oggetti

[3]Ad ogni punto corrisponde un indice biunivocamente. Il numero d'indice viene assegnato secondo un ben preciso ordine interno per cui la connettività è legata alla numerazione stessa (connettività diretta).

[4]Nella versione precedente del codice le condizioni di inlet e di outlet erano poste come condizioni iniziali ottenibili o da verifiche sperimentali o da studi sui particolari componenti, condotti di aspirazione e di scarico.

solidi che si muovono all'interno della mesh; il movimento è ottenuto tramite una tecnica di "snapper" simile a quella usata per il movimento del pistone nella versione KIVA3.

KIVA–3V usa lo stesso algoritmo risolutivo della precedente versione e risolve lo stesso set di equazioni di KIVA3 e di KIVAII.

1.2.1 Le relazioni fisiche del modello

Le equazioni utilizzate per descrivere il comportamento del fluido sono le equazioni di Navier–Stokes a cui si aggiungono le due equazioni che descrivono il modello di turbolenza, le equazioni relative alla chimica dei processi cinetici e di equilibrio, e le equazioni che governano la distribuzione dello spray.

Equazione di continuità

L'equazione di continuità della specie m

$$\frac{\partial \rho_m}{\partial t} + \nabla \cdot (\rho_m \mathbf{u}) = \nabla \cdot [\rho D \nabla (\frac{\rho_m}{\rho})] + \dot{\rho}^c_m + \dot{\rho}^s_m \delta_{ml} \tag{1.4}$$

dove ρ_m rappresenta la densità della specie m, ρ la densità totale, $\mathbf{u}$ la velocità del fluido, $\dot{\rho}^c_m$ il termine sorgente dovuto alle reazioni chimiche, $\dot{\rho}^s_m$ il termine sorgente dovuto allo spray e δ è la funzione delta di Dirac.

Equazione di conservazione della massa

La somma della relazione (1.4) per tutte le specie chimiche coinvolte fornisce l'equazione di conservazione della massa totale:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = \dot{\rho}^s \tag{1.5}$$

Conservazione della quantità di moto

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\, \mathbf{u}) = -\frac{1}{\alpha^2} \nabla p - A_0 \nabla (\frac{2}{3} \rho k) + \nabla \cdot \overline{\sigma} + \mathbf{F}^s + \rho \mathbf{g} \tag{1.6}$$

dove $\mathbf{F}^s$ la velocità di aumento del momento per unità di volume dovuta allo spray e $\mathbf{g}$ la forza di massa specifica costante. La quantità A_0 vale 0 in condizioni laminari e 1 qualora si consideri la turbolenza. $\overline{\sigma}$ rappresenta il tensore degli sforzi viscosi, ovvero

$$\overline{\sigma} = \mu \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right] + \lambda \nabla \mathbf{u}\, \overline{I} \tag{1.7}$$

dove $\overline{I}$ rappresenta la matrice identità.

Conservazione dell'energia

$$\frac{\partial (\rho I)}{\partial t} + \nabla \cdot (\rho I \mathbf{u}) = -p \nabla \cdot \mathbf{u} + (1 - A_0) \overline{\sigma} : \nabla \mathbf{u} - \nabla \cdot \mathbf{J} + A_0 \rho \epsilon + \dot{Q}^c + \dot{Q}^s \tag{1.8}$$

dove I rappresenta l'energia interna specifica, il simbolo : indica il prodotto matriciale, $\dot{Q}^c$ e $\dot{Q}^s$ sono i termini sorgenti dovuti al rilascio di calore a seguito delle reazioni chimiche ed alle interazioni con lo spray, $\mathbf{J}$ è il vettore di flusso

termico

$$\mathbf{J} = -k\,\nabla T - \rho D \sum_m \left[h_m \left(\frac{\rho_m}{\rho} \right) \right] \tag{1.9}$$

dove T è la temperatura del fluido e ρ_m l'entalpia specifica della specie m.

Relazioni di turbolenza

Il modello di turbolenza utilizzato è il k–ϵ, descritto dalle seguenti relazioni:

$$\frac{\partial(\rho k)}{\partial t} + \nabla\cdot(\rho k\mathbf{u}) = \frac{2}{3}\rho k \nabla\cdot\mathbf{u} + \overline{\sigma} : \nabla\mathbf{u} + \nabla\cdot\left[\left(\frac{\mu}{Pr_k}\nabla k\right)\right] - \rho\epsilon + \dot{W}^s \tag{1.10}$$

$$\frac{\partial(\rho\epsilon)}{\partial t} + \nabla\cdot(\rho\epsilon\mathbf{u}) = -\left(\frac{2}{3}c_{\epsilon_1} - c_{\epsilon_3}\right)\rho\epsilon\nabla\cdot\mathbf{u} + \nabla\cdot\left[\left(\frac{\mu}{Pr_\epsilon}\nabla\epsilon\right)\right] + \\ + \frac{\epsilon}{k}\left[c_{\epsilon_1}\overline{\sigma} : \nabla\mathbf{u} - c_{\epsilon_2}\rho\epsilon + c_s\dot{W}^s\right] \tag{1.11}$$

I valori delle costanti utilizzate nel modello sono:

$$c_{\epsilon_1} = 1.44 \qquad c_s = 1.5$$

$$c_{\epsilon_2} = 1.92 \qquad Pr_k = 1.0$$

$$c_{\epsilon_3} = -1.0 \qquad Pr_\epsilon = 1.3$$

Relazioni di stato

Le relazioni di stato utilizzate sono quelle per una miscela di gas perfetti:

$$p = R_0 T \sum_m \left(\frac{\rho_m}{W_m}\right) \tag{1.12}$$

$$I(t) = \sum_m \left(\frac{\rho_m}{\rho}\right) I_{pm}(T) \tag{1.13}$$

$$c_p(t) = \sum_m \left(\frac{\rho_m}{\rho}\right) c_{pm}(T) \tag{1.14}$$

$$h_m(T) = I_m(T) + \frac{R_0 T}{W_m} \tag{1.15}$$

dove R_0 è la costante universale dei gas, W_m il peso molecolare della specie m, c_{pm} il calore specifico a pressione costante della specie m.

Equazioni per la modellazione della chimica

Le equazioni utilizzate per modellare la chimica del fenomeno sono schematizzate come segue:

$$\sum_m a_{mr} x_m \leftrightarrow \sum_m b_{mr} x_m \tag{1.16}$$

dove x_{mr} è una mole della specie m e a_{mr}, b_{mr} sono i coefficienti stechiometrici della reazione r. Le reazioni chimiche sono divise in due classi, veloce e lenta, a seconda che raggiungano o meno l'equilibrio rispetto alla scala di tempo fissata dal passo di integrazione dt. La reazione cinetica r si sviluppa con velocità $\dot{\omega}_r$

$$\dot{\omega}_r = k_{fr} \prod_m \left(\frac{\rho_m}{W_m}\right)^{a'_{mr}} - k_{br} \prod_m \left(\frac{\rho_m}{W_m}\right)^{b'_{mr}} \tag{1.17}$$

I coefficienti k_{fr} e k_{br} sono definiti dalla relazione di Arrhenius:

$$k_0 = ATe^{\frac{E_a}{RT}} \tag{1.18}$$

dove E_a è l'energia di attivazione. Per una reazione all'equilibrio vale invece la relazione

$$\prod_m \left(\frac{\rho_m}{W_m}\right)^{(b_{mr}-a_{mr})} = {k_c}^r(T) \tag{1.19}$$

dove ${k_c}^r(T)$ è la costante di equilibrio, funzione della temperatura T.

Nota la velocità $\dot{\omega}_r$ delle relazioni cinetiche ci calcolano i termini sorgente nell'equazione di continuità

$${\dot{\rho}_m}^c = W_m \sum_r (b_{mr} - a_{mr})\ \dot{\omega}_r \tag{1.20}$$

ed il rilascio di calore chimico nell'equazione dell'energia

$$\dot{Q}_c = \sum_r Q_r \dot{\omega}_r \tag{1.21}$$

dove

$$Q_r = \sum_m \left[(a_{mr} - b_{mr})\ \left(\delta h_f^0\right)_m\right] \tag{1.22}$$

con δh_f^0 l'entalpia di formazione della specie m.

Equazione dello spray

Notevolmente complesso è lo sviluppo dello spray che con la miscela gassosa scambia massa, quantità di moto ed energia. Molti sono i fenomeni fisici che intervengono, come la collisione delle gocce, la loro coalescenza, oscillazione, distorsione e break–up. Per affrontare il problema si è adottata la formulazione dell'equazione dello spray che fa riferimento alla funzione di distribuzione

della goccia definita come

$$f\left(x, v, r, T_d, y, \dot{y}, t\right)\, dv\, dr\, dT_d\, dy\, d\dot{y} \tag{1.23}$$

che esprime il numero probabile di gocce per unità di volume nella posizione x ed al tempo t, con velocità nell'intervallo $(\mathbf{v}, \mathbf{v} + d\mathbf{v})$, raggio nell'intervallo $(r, r + dr)$, temperatura nell'intervallo $(T_d, T_d + dT_d)$ e parametri della distorsione negli intervalli $(y, y + dy)$ e $(\dot{y}, \dot{y} + d\dot{y})$.

1.2.2 La schematizzazione numerica

KIVA–3V risolve le equazioni che governano il moto con una approssimazione alle differenze finite. Le equazioni sono discretizzate sia nel tempo che nello spazio. L'approssimazione temporale è svolta rispetto al passo d'integrazione Δt e per una generica grandezza f vale

$$\frac{\partial f}{\partial t} = \frac{f^{n+1} - f^n}{\Delta t} \tag{1.24}$$

Metodo ALE

La differenziazione spaziale è basata sul metodo ALE (Arbitrary Lagrangian–Eulerian method). Con questo metodo la regione di calcolo è suddivisa in un insieme di celle esaedriche. I vertici della mesh si possono muovere con il fluido (modello Lagrangiano), si possono tenere fissi (modello Euleriano) oppure si possono spostare con un modello prefissato. La mesh modella la geometria

del dominio computazionale e si può spostare per seguire i cambiamenti di forma della camera di combustione. In questa schematizzazione le grandezze cinetiche (u, v, w) della miscela fluida sono riferite ad ogni nodo mentre le grandezze termodinamiche (pressione p, densità massica ρ, energia interna specifica I) sono riferite alla cella come grandezze medie sulla cella stessa. A causa di questa differenza è necessario definire una cella ausiliaria intorno a ciascun vertice, detta '*cella per la quantità di moto*', costituita dalle otto celle regolari posizionate intorno al vertice comune. Questa cella costituisce il volume di controllo d'integrazione della (1.6). L'integrazione delle equazioni (1.5) e (1.8) utilizza come volume di controllo la cella regolare.

Nel metodo ALE, il passo di integrazione durante l'intervallo di tempo prevede tre fasi di calcolo. Durante la prima fase si calcolano esplicitamente sia la velocità di ogni nodo in base ai gradienti di pressione, alle forze d'inerzia e alle forze viscose, sia l'energia interna legata al lavoro effettuato da tali forze ad esclusione di quelle di pressione che sono tenute in conto nella fase successiva. Nella seconda fase, detta Lagrangiana o diffusiva, un processo iterativo garantisce la congruenza tra il gradiente di pressione con il campo di velocità calcolato. Nella terza fase, detta *rezoning*, i vertici della mesh di calcolo, spostati durante le prime due fasi, vengono riportati nelle loro posizioni originali, tenendo in conto lo scambio di materia tra il fluido e le celle che circondano i vertici, dando così il contributo convettivo alla soluzione

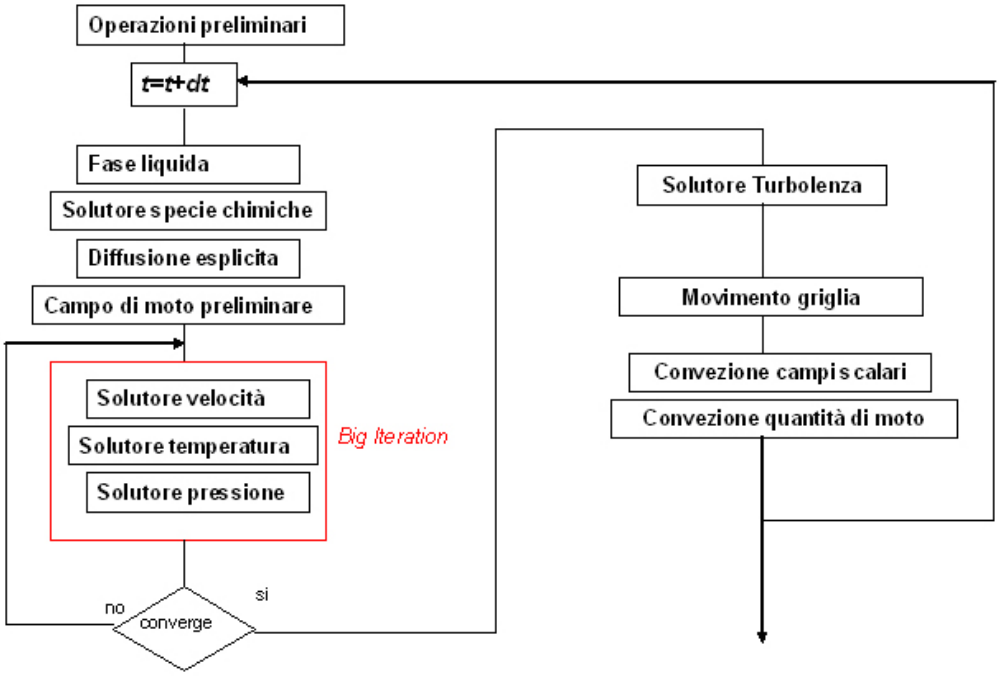


Figura 1.1: Struttura del codice di calcolo KIVA–3V

del problema.

La struttura generale del codice e le operazioni eseguite ad ogni iterazione sono riportate in figura 1.1.

Metodo DDM

Per rappresentare lo spray viene seguito il metodo DDM (Discret Droplet Model) secondo il quale un certo numero di goccioline fisiche sono raggruppate in particelle computazionali aventi medesime caratteristiche cinematiche, termodinamiche e di deformazione. L'interazione della goccia con la miscela

ai fini termodinamici avviene nella cella regolare, ai fini dello scambi della quantità di moto è invece nella *cella quantità di moto*. L'interazione del moto della particella è puramente lagrangiano:

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \mathbf{v}_p^n \, \Delta t \tag{1.25}$$

dove $\mathbf{v}^n$ viene calcolata durante la seconda fase del passo precedente.Lo spostamento, la collisione, l'evaporazione, e la distruzione delle gocce sono calcolati durante la prima fase. La terza fase di rezone lascia immutate le particelle.

1.2.3 La modellazione delle valvole in KIVA–3V

Come si è accennato in precedenza il miglioramento sostanziale di KIVA–3V rispetto alle precedenti versioni è l'effettiva capacità di modellare le valvole di aspirazione e di scarico ed il loro movimento all'interno del dominio. KIVA–3V è in grado di modellare qualsiasi numero di valvole sulla testa del cilindro. Ogni valvola può avere dimensione e profilo differente, accuratezza nella discretizzazione appropriata al grado di finezza dei risultati che si vuole ottenere, e propria legge di alzata. Inoltre la valvola può essere montata indifferentemente in posizione verticale, con l'asse della valvola parallelo all'asse del cilindro, o inclinata rispetto allo stesso. L'inclinazione è però permessa esclusivamente nel piano xz. Nello spazio logico la valvola si può muove-

re esclusivamente nelle direzione 'bottom–top' (`z`). Non è possibile quindi analizzare né valvole montate orizzontalmente né anulari.

Si è altresì detto come la valvola venga considerata dal codice come un oggetto solido immerso ed in movimento all'interno della griglia. La tecnica con la quale viene fatta muovere la valvola è molto simile a quella con la quale viene fatto muovere il pistone. Mentre la subroutine `SNAPB` assolve al compito di muovere il pistone, le subroutine `SNAPVFACE` e `SNAPVTOP` sono quelle delegate al movimento rispettivamente della superficie inferiore che si affaccia verso il cilindro la prima, e la superficie superiore che si affaccia verso il condotto la seconda. Per far posto all'inclinazione delle valvole queste nuove subroutine includono il movimento di snapper non solamente lungo la direzione z ma anche lungo la direzione x.

Così come l'originale KIVA3, la griglia fornita al codice come dato di input nel file `itape17` deve essere tale che corrisponda alla geometria del pistone quando questo si trova nel punto morto inferiore (`BDC`). Insieme ai dati geometrici sulla mesh deve essere fornita al codice anche una tabella contenente una serie di flag che permette di individuare il tipo di nodo o di cella su cui si sta lavorando. In aggiunta alle sei flag presenti anche nel KIVA3, KIVA–3V necessita di una settima flag, chiamata *idface*. Questa è una flag di nodo, ossia associata direttamente ai vertici della mesh, e permette l'individuazione del tipo di movimento a cui deve essere sottoposto il nodo

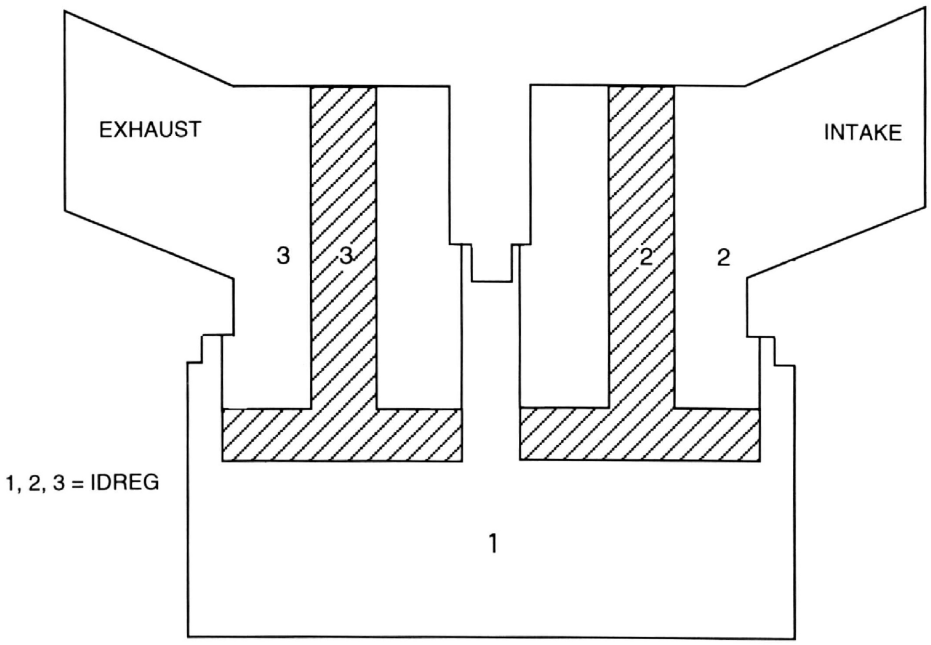


Figura 1.2: Assegnazione del flag *idreg* in diverse zone del dominio

stesso.

Nel dominio di calcolo devono essere presenti tipicamente tre regioni fisiche (fig. 1.2). Queste tre regioni di calcolo devono essere distinte nella griglia di calcolo attraverso una opportuna flag '`idreg`'. Alle celle interne alla zona del cilindro viene assegnato `idreg`=1. `idreg`=2 può essere riferita al collettore di aspirazione mentre `idreg`=3 a quello di scarico, o viceversa. L'unica ambiguità possibile che si può presentare è durante l'alzata valvola, poiché, contrariamente a quanto accade quando le valvole sono in posizione di chiusura, non si presenta più soluzione di continuità tra le diverse regioni. In

questo caso, le celle che topologicamente si trovano al di sopra della valvola presenteranno un `idreg` pari al valore dell'`idreg` della valvola stessa, mentre quelle al di sotto e laterali avranno `idreg`=1. Durante il calcolo le subroutine `SNAPVFACE` e `SNAPVTOP` inseriscono e disattivano file di celle al di sopra e al di sotto della valvola; a seconda che la stessa si sposti verso l'alto o verso il basso, automaticamente il codice assegna a queste file di celle nuove un valore di `idreg` appropriato.

Il nuovo vettore di flag, `idface`, deve essere fornito dal generatore di griglia come parte del file itape17[5]. Nel KIVA3 erano possibili esclusivamente due tipo di superfici mobili: il pistone ed il secondo pistone in testa opzionale. Il codice era in grado di identificare le superfici mobili ed il tipo di superficie mobile attraverso le coordinate `z` e le informazione derivanti dalle flag già presenti nel codice, senza quindi la necessità di dati di input addizionali. L'implementazione delle valvole e la possibilità di effettuare alzate diverse per ogni valvola porta alla necessità di informazioni addizionali per identificare quale superficie mobile si sta processando. Data questa nuova definizione, il pistone è sempre identificato come superficie mobile 0, e quindi ogni vertice appartenente alla superficie del pistone ha un `idface`=0. I vertici

[5]L'aggiunta dell'idface è l'unica differenza che sussiste tra il file itape17 ed il file contenente i dati di input per KIVA3 tape17

invece appartenenti al pistone contrapposto in testa sono identificati da un `idface`=1.

Sebbene sia la superficie superiore che quella inferiore della stessa valvola si muovano con la medesima velocità, ciascuna è identificata separatamente in quanto ogni superficie mobile è trattata come un entità a se stante dallo snapper della valvola. Poiché un secondo pistone in testa non è un opzione compatibile con la presenza delle valvole, l'`idface`=1 è un valore disponibile. Allora i vertici delle superfici *bottom* della valvola saranno identificati da un valore dispari dell'`idface` (1,3,5,...) a le superfici top della valvola saranno identificate da un valore pari dell'`idface` (2,4,...). Attraverso questa definizione, ogni superficie in movimento, che sia appartenente ad un pistone o che sia appartenente ad una valvola, se ha delle celle fluide al di sopra di essa ha un `idface` pari, al contrario ogni superficie in movimento che ha al di sotto celle fluide è identificata da un `idface` dispari. A tutti i vertici restanti, che non sono associati a nessun tipo di superficie mobile, viene assegnato un `idface`=-1.

I dati relativi alle valvole presenti all'interno del dominio viene comunicato al codice all'interno del file itape5[6]. In particolare la variabile `nvalves`

[6]Il file itape5 per KIVA–3V corrisponde, a meno di differenze introdotte dovute alla presenza delle valvole, al file di input itape della precedente versione KIVA3.

corrisponde al numero di valvole presenti; per ogni valvola occorre specificare: `vliftmin` la minima alzata in cm al di sotto della quale la valvola viene considerata a livello di calcolo chiusa, `skirt` lo spessore, sempre in cm, del bordo estremo verticale della valvola; `tmove`[7] la temperatura, espressa in Kelvin, della valvola; `vtiltxz` l'inclinazione dell'asse della valvola, in un piano parallelo al piano xz, rispetto all'asse del cilindro, espressa in gradi; `nlift` il numero dei dati di alzata delle valvole presente nel file `itape18`.

Il file itape18[8] è un file di testo formattato, scritto con un formato libero, che contiene la tabella dell'alzata delle valvole. Delle due colonne di cui è composto, la prima identifica l'angolo di manovella[9] e la seconda i corrispondenti valori dell'alzata valvola espressi in cm[10]. La tabella deve essere provvista di informazioni relative ad un ciclo completo, da 0° a 720° per un motore 4 tempi, da 0° a 360° per un motore 2 tempi. Il codice determina la lunghezza di un intero ciclo del motore attraverso la variabile di input `revrep` (revolution between repetion), la quale deve essere specificata pari a 1 per i motori 2 tempi, pari a 2 per i motori 4 tempi.

[7]Poiché è ora possibile specificare per ogni valvola il valore della temperatura, la linea nella precedente versione dell'itape5 relativa a tvalve è stata in questa versione eliminata.

[8]Questo file non era presente in nessuna versione precedente di KIVA.

[9]I valori sono degli interi.

[10]Le alzate sono espresse attraverso dei numeri reali.

Tutti gli angoli di manovella corrispondenti ad una alzata nulla possono essere eliminati dal file, in modo da minimizzarne la lunghezza. Inoltre, l'incremento dell'angolo tra un riga e la successiva non è richiesto che sia uniforme all'interno del file. L'andamento dell'alzata per ogni valvola appare in successione all'interno del file, in particolare le prime `nlift` linee corrisponderanno alla valvola numero 1[11], le seconde `nlift` alla seconda e così via.

1.2.4 Movimento delle valvole

Per ogni ciclo, la subroutine `VALVE` interpola i dati presenti nel file `itape18` e da questi determina la velocità istantanea di ogni valvola. L'angolo di manovella della viene normalizzato, cioè riportato nell'intervallo [0:720] ([0:360] per motori due tempi), permettendo così al codice di effettuare cicli multipli del motore.

Nello spazio fisico, i vertici che giacciono sulla superficie mobile[12] della valvola vengono spostati rigidamente ad ogni ciclo nella subroutine `REZONE`,

[11] La numerazione delle valvole è tale per cui la prima valvola è quella con idface 1 e 2, la seconda ha idface 3 e 4 e così via.

[12] Lo stelo della valvola non è una superficie mobile, o quantomeno non subisce *snapping*; la sua flag idface è comunque pari a 2, in quanto ad essa è associata una velocità diversa da zero.

usando la velocità corrente della valvola determinata in precedenza. Anche se fisicamente i vertici appartenenti allo stelo valvola non si muovono all'interno del dominio, comunque a questa superficie viene associata una velocità pari a quella della valvola in modo tale da poter calcolare gli sforzi viscosi tra fluido e stelo stesso e permettere, nel caso di iniezione nei condotti, alle gocce depositatesi sullo stelo stesso di muoversi con la valvola. Per preservare la griglia originaria nei condotti, questi vertici non vengono mai spostati dalla subroutine `REZONE`.

1.2.5 *Snapping* della griglia

Nelle griglie che presentano valvole verticali lo snapping[13] delle valvole è trattato in maniera del tutto analoga allo snapping del pistone, ossia la superficie della valvola subisce uno snapping quando la superficie stessa si è mossa di una quantità pari alla metà della distanza iniziale tra la superficie stessa e i vertici subito sopra o sotto di essa, a seconda del tipo di movimento della valvola.

Nei domini che presentano valvole inclinate rispetto all'asse del cilindro, la griglia può essere molto distorta in prossimità della valvola, e comunque

[13]Con il termine *snapping* si intende una variazione topologica della griglia di calcolo che aggiunge o toglie piani di celle in prossimità delle superfici mobili, così da consentire ampi movimenti delle componenti del motore.

non è uniforme. Non è quindi possibile determinare una dimensione di riferimento per effettuare lo snapping. In questo caso viene definita una tabella in cui la massima alzata viene suddivisa in sei frazioni, anche non uniformi. Questa tabella viene utilizzata per determinare l'istante in cui effettuare lo snapping. Questo approccio richiede che le linee di griglia immediatamente al di sotto della faccia inferiore della valvola debbano essere piuttosto vicine alla faccia stessa della valvola al fine di evitare un improvviso e vistoso cambiamento delle dimensioni delle celle quando si effettua lo snapping, che potrebbe portare ad errori geometrici nella definizione delle celle stesse[14].

Le superfici superiore ed inferiore della valvola vengono spostate da due routine dedicate. Durante questo processo, la valvola può avere temporaneamente una altezza pari a due celle, anche se le sue dimensioni geometriche non vengono mai cambiate. Si consideri, ad esempio, un evento di snapping mentre la valvola si sposta verso il basso. Se la superficie superiore effettua il cambiamento topologico prima di quella inferiore, le due superfici si troverebbero a condividere gli stessi nodi, ma geometricamente devono trovarsi distanti fra loro. Se invece la superficie inferiore effettua lo *snapping* prima della superiore, la valvola avrà due celle tra le due superfici, ma la coerenza

[14]Si parla in questo caso di *inversione* di cella. Questo avviene quando la posizione spaziale dei suoi vertici non corrisponde al loro ordine nella sua definizione topologica. Il calcolo del volume di una cella *invertita* restituisce un valore negativo

tra la struttura topologica e la geometria sarà, in ogni momento, rispettata.

Quando la valvola è costituita da due celle, mentre i vertici al suo interno vengono deattivati, il valore dell'`idface` sulla superficie laterale (*skirt*). Poiché questi vertici non appartengono né alla superficie bottom né ad una superficie top della valvola, non richiedono un particolare trattamento durante lo snapping della valvola, l'unico vincolo è il loro posizionamento geometrico. La convenzione utilizzata, ovviamente arbitraria, è quella di assegnare a questi vertici lo stesso `idface` caratteristico dei vertici immediatamente al di sotto. Ovviamente tutto questo si ripercuote come vincolo sulla generazione della mesh. Non è infatti compatibile con il codice una mesh che abbia il corpo della valvola con una suddivisione dello skirt superiore a due celle.

1.2.6 Chiusura ed apertura delle valvole

L'ultimo snap della griglia, durante la fase di chiusura, porta ad un minimo di una cella di ingombro tra la testa della valvola e la sede della stessa. Nel caso in cui la valvola chiudesse successivamente all'ultimo evento di snapping, l'unica cella rimasta continuerebbe a collassare su se stessa. Se fosse permesso alla valvola di appoggiarsi completamente alla sede, ne risulterebbe uno spessore finale delle celle nullo, situazione non ammissibile nei metodi ai volumi finiti. Per prevenire questa situazione, la chiusura della valvola viene

fatta terminare nel momento in cui è raggiunto un ingombro minimo specificato in ingresso, tipicamente compreso tra 0.20 e 0.50mm. La chiusura della valvola, nel senso di separazione del volme nel cilindro dal volume nel condotto, viene simulata attraverso una modificazione delle flag di faccia delle celle al di sopra della valvola. In particolare vengono modificate dal codice le flag di BC (*Boundary Conditions*) della corona di facce[15] delle celle intorno alla periferia della valvola, zona denominata di 'spiraglio'. Le BCL e BCF di queste celle vengono modificate da una condizione di parete fluida ad una di parete solida[16], il che impedisce al fluido di spostarsi dal cilindro al condotto e viceversa.

Durante l'apertura la procedura è analoga, ma inversa, a quanto descritto sopra. Le condizioni di superficie solida vengono convertite in fluide quando la valvola raggiunge l'altezza specificata, e la connessione tra condotto e cilindro viene ripristinata.

È facile notare come questa strategia introduca alcuni errori. La chiusura e l'apertura effettive della valvola sono sfasate di un certo intervallo di tempo, rispetto alle condizioni geometriche. Questi eventi, inoltre, avvengono improvvisamente, ovvero l'area di passaggio cambia da 0 ad un valore

[15] *left*, *right*, *front* e *derriere*, nella nomenclatura utilizzata da KIVA

[16] È questa l'unica condizione in cui KIVA ammette una parete solida tra due celle fluide.

significativamente maggiore in un solo intervallo di calcolo. Il fatto, però, che al momento della chiusura valvola, così come all'apertura, l'efflusso sia molto piccolo, se confrontato a quanto misurato alle massime alzate, rende trascurabili gli effetti di queste approssimazioni sull'intera fase di aspirazione o di scarico.

Occorre notare che ridurre molto il valore di alzata al di sotto del quale la valvola viene chiusa non porta apprezzabili miglioramenti ai fini della soluzione, ma comporta una considerevole riduzione del time step di integrazione, a causa delle elevate velocità che si raggiungono nella sezione di efflusso[17].

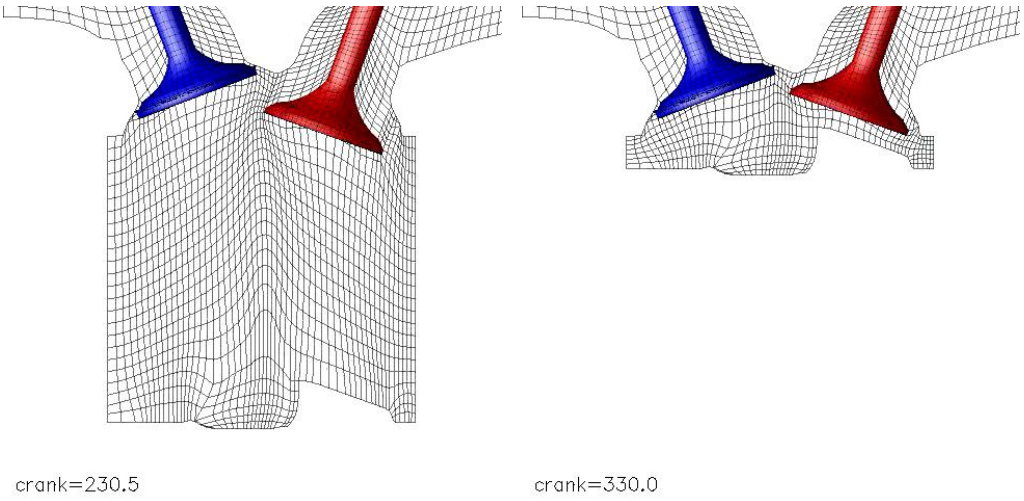
Anche dopo la chiusura "numerica", KIVA–3V continua comunque a seguire il vero andamento dell'alzata valvola fino all'angolo relativo alla chiusura fisica. L'alzata reale viene utilizzata dal modello di 'film liquido' e consente alle particelle di combustibile depositatesi in forma liquida sul bordo della valvola di scivolare all'interno della camera per tutto il tempo a disposizione, fenomeno questo particolarmente importante nelle applicazioni del codice a motori ad iniezione nei condotti di aspirazione.

[17]Si consideri ad esempio la fase di apertura della valvola di scarico, quando il gradiente di pressione tra cilindro e condotto è molto alto.

1.3 Rigenerazione dinamica della griglia

Nel corso della simulazione, in particolar modo per motori con valvole inclinate rispetto all'asse del cilindro, i soli eventi di *snapping* non sono in grado di preservare la validità della griglia nel corso di un intero ciclo, rendendo quindi necessaria una strategia di riposizionamento dei nodi della griglia in grado di gestire geometrie come in figura 1.3. In KIVA–3V non è presente una versione definitiva e valida per ogni geometria di un algoritmo per la rigenerazione dinamica della griglia di calcolo. Le stesse note di rilascio del codice specificano che le tre subroutine predisposte al questo compito, `REZWEDGE`, `REZPENT` e `REZCOMB`, ognuna destinata ad una diversa tipologia di camera di combustione, sono comunque tali da poter essere adattate esclusivamente ai *test case* da loro stessi effettuati.

Nel corso del presente lavoro, e di altri che lo hanno preceduto, l'algoritmo di rigenerazione della griglia è stato completamente rivisto, sia dal punto di vista dell'implementazione, sia da quello dell'utilizzazione, essendo ora completamente controllabile da file di ingresso, al variare sia delle zone che di diversi intervalli di angoli di manovella. Per una più esauriente descrizione di quanto realizzato si rimanda al capitolo 2.



(a) In prossimità del PMI (b) In prossimità del PMS

Figura 1.3: Esempio di deformazione della mesh

Capitolo 2

Sviluppo e parallelizzazione di KIVA–3V

Il costante aumento della dimensione delle griglie di calcolo, legato alla necessità di simulare sistemi sempre più complessi da un lato, e di ricercare soluzioni più accurate dall'altro, e la sempre maggiore importanza che la fluidodinamica computazionale riveste in molti ambiti industriali hanno fatto si che l'efficienza di un codice di simulazione sia uno dei fattori chiave per la sua effettiva utilizzazione.

Kiva–3V è stato quindi profondamente rivisto [10, 16, 17, 19–23], con lo scopo di ottimizzarne l'efficienza, razionalizzare l'uso delle risorse impiegate ed estenderne le funzionalità. I passi seguiti durante questo processo sono riassumibili come segue:

- Sviluppo del codice seriale
- Parallelizzazione
- Sviluppo di nuove funzionalità

2.1 Sviluppo del codice seriale

Per quanto il presente lavoro riguardi principalmente la parallelizzazione e lo sviluppo di nuove funzionalità, è opportuno accennare brevemente alle modifiche apportate a Kiva–3V come codice di calcolo seriale, senza le quali la parallelizzazione non sarebbe stata possibile e l'introduzione di nuove caratteristiche difficile e poco utile.

In primo luogo, il codice sorgente, scritto in FORTRAN77, è stato interamente rivisto eliminando la dichiarazione implicita delle variabili e i blocchi `COMMON` ed adottando uno stile più vicino allo standard Fortran95. Questo ha permesso di evitare errori comuni nello stile precedente, dal semplice errore di battitura fino all'accesso a zone di memoria non lecite, ed ha altresì consentito di utilizzare gli strumenti di ottimizzazione e di diagnostica messi a disposizione dai moderni compilatori. La dichiarazione esplicita delle variabili e l'utilizzo dei *moduli* al posto dei blocchi `COMMON` ha inoltre reso il codice più razionale e leggibile.

Un'altra importante modifica è stato l'utilizzo di variabili allocate dinamicamente, al posto della loro dichiarazione statica. Per quanto sia una caratteristica comune nei moderni linguaggi di programmazione, questo non era possibile in FORTRAN77 ed è stato introdotto solo a partire dallo standard Fortran90. In questo modo un solo eseguibile è in grado di gestire qualsiasi dimensione di griglia, con l'unico limite della memoria fisica disponibile, senza dover essere ricompilato, e senza sprechi di risorse. La gestione dello spray di combustibile diventa molto più efficiente, poiché il numero di gocce varia notevolmente nel corso della simulazione. È opportuno notare che con i primi compilatori Fortran90 utilizzati si registrava un netto calo di prestazioni con l'utilizzo di variabili dinamiche, probabilmente dovuto ad inutili copie durante il passaggio di tali variabili a delle subroutine. Questo problema è stato via via risolto nelle varie versioni di compilatori che sono state rilasciate nel tempo, consentendo infine l'utilizzo di variabili dinamiche senza alcuna penalizzazione nelle prestazioni.

Sono stati introdotti nuovi solutori per la fase implicita ed esplicita che, a differenza dei precedenti, prevedono l'assemblaggio (esplicito o meno) di una matrice di coefficienti per le equazioni degli scalari e delle velocità basati sulle librerie `BLAS`[1] [11–15]. Attorno ad esse è stata costruita una struttura

[1]Acronimo per Basic Linear Algebra Subprograms. Si tratta di un insieme di routine che forniscono degli strumenti per effettuare in maniera efficiente operazione basiche per

in grado di adattarsi in maniera molto efficiente al modello di gestione della griglia di calcolo su cui Kiva–3V è basato. L'introduzione di questa nuova tipologia di solutori ha il duplice vantaggio di ridurre drasticamente il tempo di calcolo necessario per la soluzione delle equazioni e, allo stesso tempo, di adattarsi molto bene alla successiva fase di parallelizzazione.

In breve, in un problema la cui discretizzazione viene effettuata utilizzando il metodo dei volumi finiti[2], il fenomeno viene rappresentato da operatori differenziali, quindi locali. Ogni volume in esame, quindi, risente solo di un certo numero di volumi vicini, numero dipendente dall'ordine dello schema di discretizzazione e dalla topologia della griglia di calcolo, ma non dalla dimensione del problema. Il numero di questi volumi vicini corrisponde al numero di elementi non nulli nella riga della matrice dei coefficienti del sistema di equazioni corrispondente al volume di controllo in esame. La matrice $N \times N$ dei coefficienti, essendo N il numero dei volumi di controllo, ha quindi le seguenti caratteristiche:

- Numero di coefficienti non nulli molto piccolo rispetto ad N

vettori e matrici. I produttori di CPU rilasciano versioni delle librerie ottimizzate per i vari processori, l'utilizzo delle quali migliora sensibilmente le prestazioni di ogni applicazione scritta utilizzando le BLAS.

[2]Lo stesso avviene quando la discretizzazione avviene alle differenze finite o agli elementi finiti.

- Numero di coefficienti non nulli indipendente da N, ma legato solamente alla topologia del problema
- Coefficienti non nulli localizzati prevalentemente nei dintorni della diagonale principale

Questa tipologia di matrici va sotto il generico nome di "matrici sparse" e per esse esistono algoritmi di risoluzione dedicati molto efficienti. Sono inoltre molto vantaggiose dal punto di vista della memoria, in quanto non è necessario memorizzare l'intera matrice, ma solo gli elementi non nulli, insieme alla loro posizione all'interno della matrice. Viene quindi utilizzata una libreria dedicata, chiamata `PSBLAS`[3] [16], che è in grado di risolvere in maniera efficiente sistemi di equazioni lineari rappresentati da matrici sparse, partendo dal nucleo delle `BLAS`, anche in regime di calcolo parallelo.

Il codice è stato poi rivisto in alcuni punti particolarmente dispendiosi in termini di tempo di calcolo, come il posizionamento iniziale della griglia ed il calcolo dei flussi convettivi. In particolare per questi ultimi è stata ottenuta una riduzione del tempo di calcolo pari a circa il 30%, utilizzando prevalentemente alcune accortezze nella scrittura del sorgente in maniera da far generare codice più efficiente al compilatore, piuttosto che intervenendo sull'algoritmo stesso.

[3] Acronimo per Parallel Sparse BLAS

La scelta dei parametri di convergenza dei solutori, così come della necessità o meno di precondizionare le matrici ed eventualmente il tipo di precondizionatore da utilizzare è stata oggetto di analisi, cercando per ogni variabile un opportuno compresso tra prestazioni ed accuratezza della soluzione.

Infine, una migliore gestione degli errori è stata implementata lungo tutto il processo di inizializzazione del caso e soluzione.

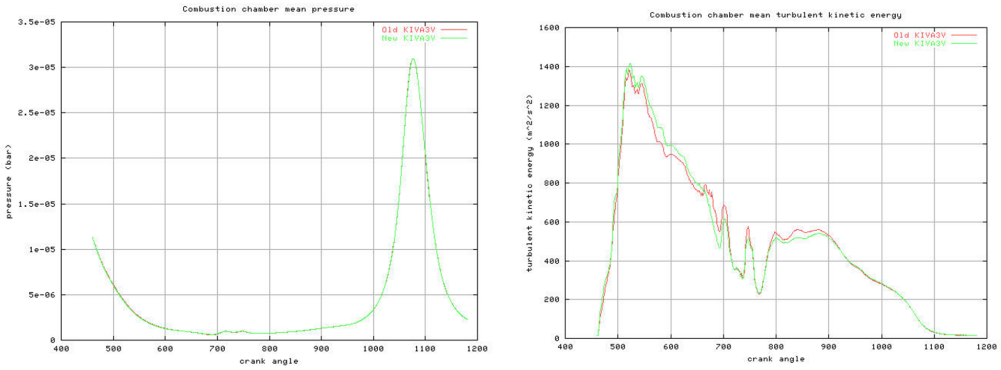
2.1.1 Validazione del codice seriale

Utilizzando come benchmark un caso test Renault Sport, con regimi di rotazione di 16000 e 13000 rpm, sono state confrontate sia grandezze termodinamiche medie all'interno del cilindro, sia grandezze puntuali misurate in alcuni punti ritenuti significativi. Il confronto tra la versione originale e la nuova mostra come non siano state introdotte modifiche significative nella soluzione fisica, con un notevole vantaggio nei tempi di calcolo. È necessario sottolineare come, affinché fossero il più possibile rappresentative del funzionamento a regime di un motore, le simulazioni siano state eseguite in un ampio arco di gradi di manovella e in presenza di iniezione.

Un confronto tra i valori medi in camera di combustione viene riportato in figura 2.1. I valori puntuali sono stati invece misurati in tre diverse posizioni, una nel condotto di aspirazione, una in camera di combustione, nella zona di

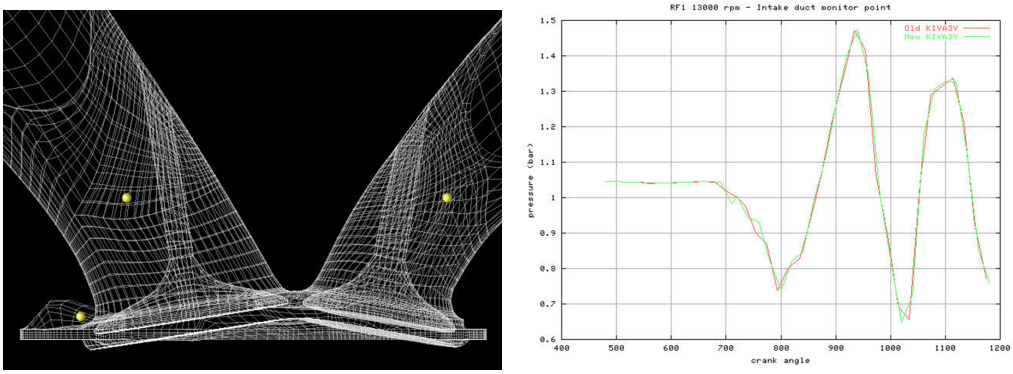
iniezione, ed il terzo nel condotto di scarico, come riportato in figura 2.2(a).

I risultati misurati vengono riportati nelle figure da 2.2(b) a 2.2(j).



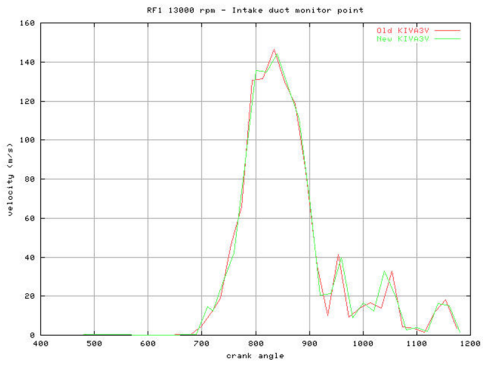
(a) Pressione media (b) Energia cinetica turbolenta media

Figura 2.1: Confronto valori medi nel cilindro al variare del solutore.

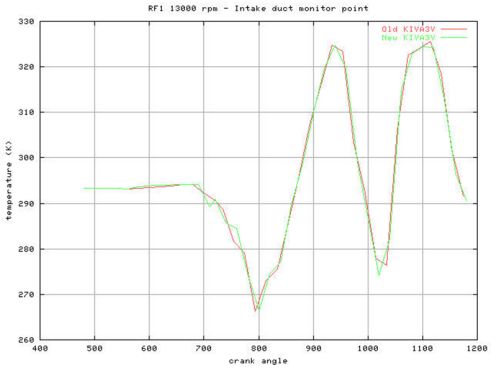


(a) Punti di misurazione (b) Aspirazione: pressione

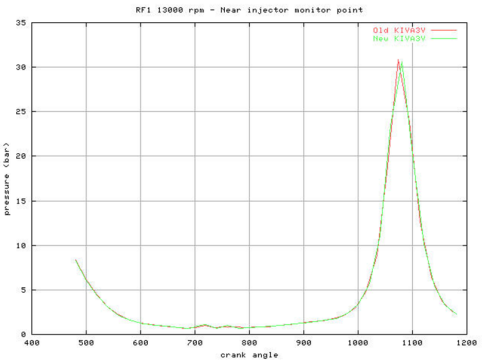
Figura 2.2: Confronto tra le variabili in esame nei punti di misurazione



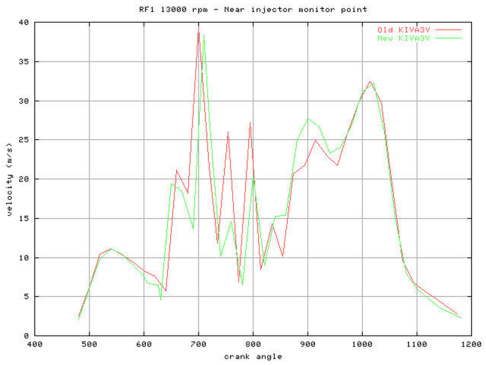
(c) Aspirazione: velocità



(d) Aspirazione: temperatura

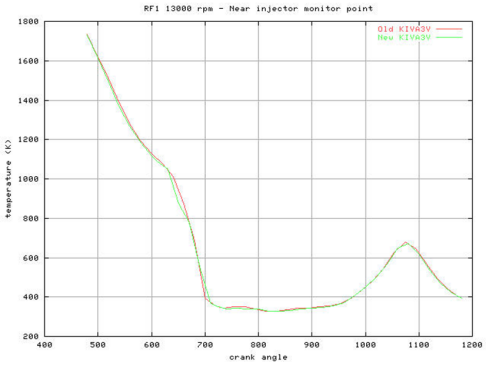


(e) Cilindro: pressione

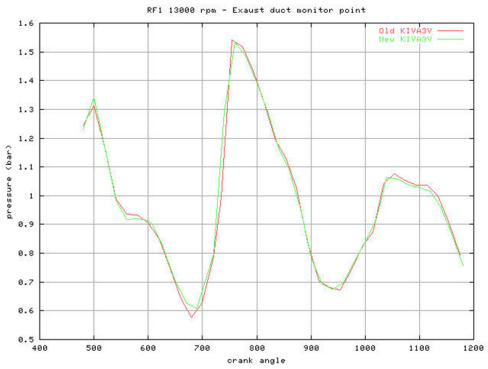


(f) Cilindro: velocità

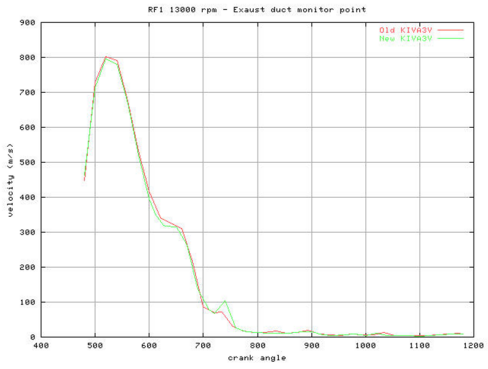
Figura 2.2: Confronto tra le variabili in esame nei punti di misurazione



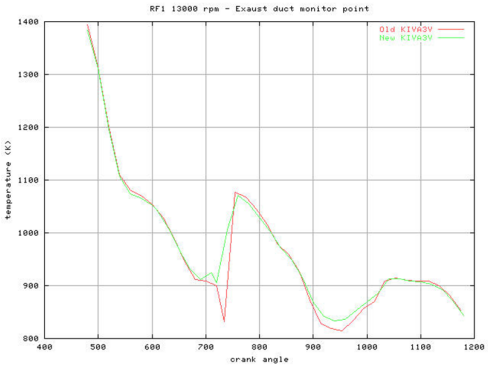
(g) Cilindro: temperatura



(h) Scarico: pressione



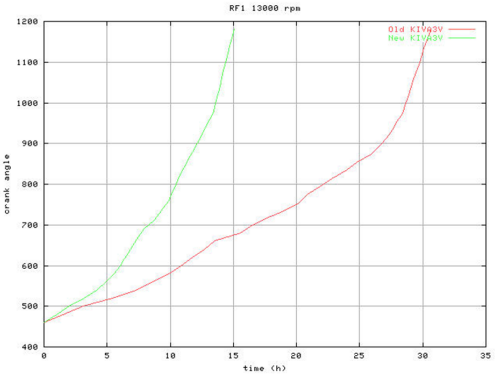
(i) Scarico: velocità



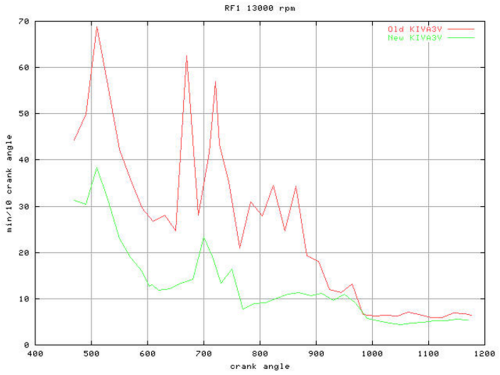
(j) Scarico: temperatura

Figura 2.2: Confronto tra le variabili in esame nei punti di misurazione

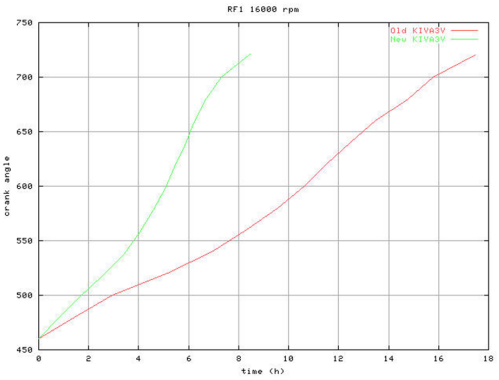
Verificato che la qualità della soluzione non è stata compromessa, rimane da confrontare i tempi di calcolo tra i diversi solutori. I grafici corrispondenti ai due diversi regimi di rotazione sono riportati nelle figure 2.3 (a), (b), (c), (d). Vengono in particolare rappresentati l'angolo di manovella raggiunto in funzione del tempo trascorso (figure 2.3(a) e 2.3(c)) e il tempo impiegato dalla simulazione per avanzare di 10 gradi, in funzione dell'angolo di manovella di simulazione raggiunto (figure 2.3(b) e (d)). Questo permette sia di valutare i miglioramenti in termini assoluti, si di analizzare in quali circostanze i nuovi solutori rappresentano un beneficio. Si può infatti dedurre che il maggior risparmio di tempo di calcolo si ha quando i gradienti di pressione nel dominio sono più alti, ovvero quando le matrici dei sistemi da risolvere, in particolare per pressione e velocità, richiedono maggiore sforzo computazionale.



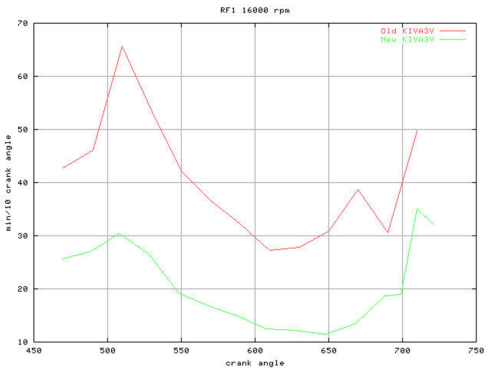
(a) Tempo di calcolo – 13000 rpm



(b) Tempo di calcolo per 10 gradi di angolo di manovella – 13000 rpm



(c) Tempo di calcolo – 16000 rpm



(d) Tempo di calcolo per 10 gradi di angolo di manovella – 16000 rpm

Figura 2.3: Confronto dei tempi di calcolo tra nuovi solutori ed originali, per diversi regimi di rotazione.

2.2 Parallelizzazione

2.2.1 Descrizione generale

Come precedentemente accennato, i nuovi solutori usati all'interno del codice, basti sulle libreria `PSBLAS`, sono stati la premessa necessaria per la parallelizzazione del codice, ma molto altro è stato necessario sviluppare. Per parallelizzazione si intende la capacità del programma di eseguire le operazioni richieste contemporaneamente su più processori, in maniera da arrivare alla soluzione in un tempo generalmente minore. Il vantaggio di un codice di calcolo paralleo rispetto ad uno seriale dipende da molteplici fattori, alcuni dei quali portano ad ulteriore benefici sul tempo di calcolo, come ad esempio l'efficienza dell'accesso alla cache dei moderni processori, altri invece ne limitano fortemente la scalabilità, come la necessità di scambiare informazioni tra i diverse processi di calcolo coinvolti.

La parallelizzazione può essere di tipo *funzionale*, in cui cioè ogni processo svolge compiti diversi, oppure può prevedere la *decomposizione del dominio* di calcolo, ed è questa la scelta obbligata per applicazioni agli elementi o ai volumi finiti, ed è quindi questa la strada che è stata seguita.

La seconda scelta riguarda lo standard di comunicazione tra i processi di calcolo, strettamente legato al tipo di architettura del computer parallelo su cui il programma verrà utilizzato. Senza scendere troppo nei dettagli, basta

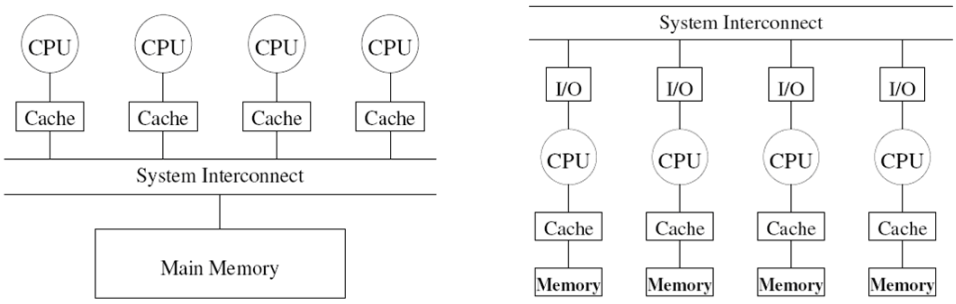
dire che gli standard come OpenMP si rivolgono ad architetture a memoria condivisa[4], mentre altri, come MPI[5], sono pensati per macchine a memoria distribuita. Quest'ultimo approccio è di gran lunga il più utilizzato attualmente, poiché i sistemi a memoria distribuita formati da vari computer più o meno "standard" collegati tra loro da reti veloci dedicate sono relativamente economici, in quanto beneficiano della riduzione dei costi propria dell'economia di scala dell'hardware per computer di largo consumo, e sono facilmente espandibili, in quanto basta virtualmente aggiungere più unità base per aumentare la capacità di calcolo. Una schematizzazione di un sistema a memoria condivisa è rappresentato in figura 2.4(a), di un sistema a memoria distribuita in figura 2.4(b).

A titolo di esempio, mentre i più grandi sistemi a memoria condivisa attualmente esistenti sono costituiti da decine di CPU, i più grandi cluster, tralasciando i sistemi MPP[6], sono costituiti da decine di migliaia di processori. Nella semestrale classifica dei sistemi di calcolo più veloci al mondo i

[4]Sono attualmente in fase di sviluppo delle interfacce tipo OpenMP ma dedicate a sistemi a memoria distribuita. L'obiettivo degli sviluppatori di questi sistemi è di unire la semplicità di scrittura del codice propria di OpenMP con i bassi costi e la scalabilità dei cluster di computer.

[5]Message Parsing Interface

[6]Massively Parallel Processing



(a) Memoria condivisa (b) Memoria distribuita

Figura 2.4: Schema di computer a memoria condivisa e distribuita

cluster di computer sono di gran lunga i più diffusi, come riportato in figura 2.5[7]. È quindi facile capire il motivo per cui tutti i maggiori software per CFD, in particolare per applicazioni industriali, sono sviluppati per cluster ed utilizzano lo standard MPI.

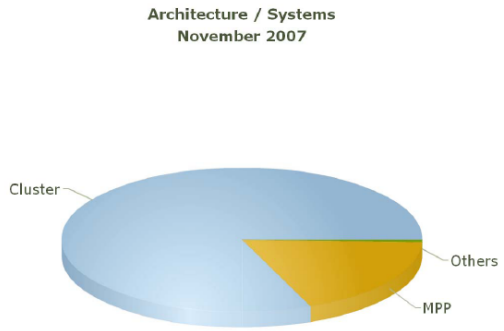


Figura 2.5: Distribuzione dei 500 maggiori supercomputer in base al tipo

[7]Fonte: Top500

La strategia di decomposizione del dominio, associata all'interfaccia MPI, che non preclude la possibilità di eseguire il programma su computer multiprocessore a memoria condivisa, si adatta bene a problemi la cui soluzione viene valutata localmente, come i metodi CFD ai volumi finiti. Ogni processore può infatti risolvere il problema nella porzione del dominio ad esso assegnata, rappresentata in colori diversi in figura 2.6, avendo bisogno di conoscere solo ciò che sta nelle immediate vicinanze dell'interfaccia con un altro processo. È opportuno notare che la forma e la dimensione della zona di comunicazione tra partizioni diverse non dipende solamente dalla geometria, ma anche dall'algoritmo di partizionamento e dal numero di processi utilizzato, come mostrato in figura 2.7

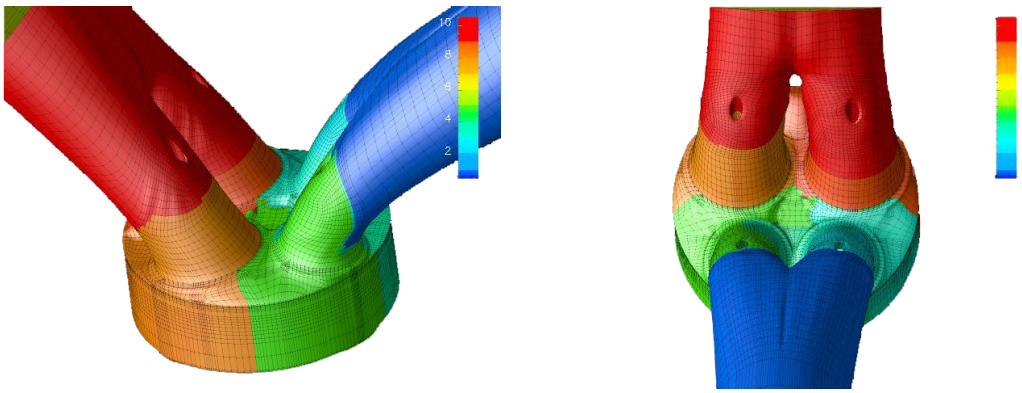
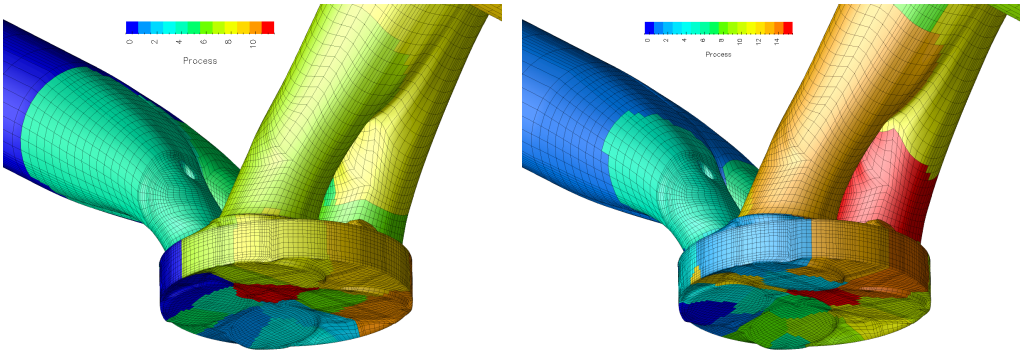


Figura 2.6: Esempio di decomposizione del dominio di calcolo

Non solo la parte di soluzione delle equazioni, però, ma la stessa struttura



(a) 12 processi (b) 16 processi

Figura 2.7: Decomposizione del dominio per diverso numero di processi

del programma ha bisogno di profonde modifiche per poter funzionare efficientemente e con tutte le normali funzionalità in parallelo. Questo permette di poter eseguire simulazioni su cluster di computer relativamente economici, se confrontati con macchine multiprocessore a memoria condivisa, raggiungendo alti livelli di prestazione e la possibilità di gestire problemi molto grandi, ovvero griglie di calcolo strutturate formate da un numero molto alto di nodi.

2.2.2 Dettagli sull'implementazione

Vengono qua di seguito brevemente descritte le caratteristiche proprie della parallelizzazione del codice seriale.

Gestione dei file di ingresso e partizionamento

Al fine di evitare problemi di sincronizzazione dei dati da leggere da disco, cosa che potrebbe essere non banale qualora ogni processo accedesse ai dati su una unità disco non condivisa con gli altri, tutti i file di ingresso vengono letti da un solo processo, detto *root*, e le informazioni in essi contenute vengono opportunamente spedite a tutti gli altri[8].

Questa scelta da un lato complica la scrittura del codice e può rappresentare una perdita di efficienza, poiché i processi non impegnati nelle operazioni di lettura rimangono in attesa che *root* termini la fase di I/O, mantenendo però occupate le risorse ad essi assegnate. D'altro canto la fase di lettura è comunque breve rispetto ai tempi caratteristici della simulazione ed avere i file di ingresso centralizzati consente modifiche ai parametri della simulazione più efficienti e veloci.

Un'attenzione particolare richiede il trattamento della mesh di calcolo. Questa viene letta dal processo *root* ma non deve essere spedita interamente agli altri. Ogni processo ha infatti necessità di conoscere la propria parte di dominio più alcune informazioni nella zone che confinano con altri processori.

[8]Nel caso in cui l'esecuzione sia seriale, le routine di spedizione non inviano effettivamente dati, ma lasciano comunque libero il processo principale di continuare. La simulazione può quindi proseguire senza problemi.

La decomposizione del dominio, di cui un esempio è mostrato in figura 2.6, può essere effettuata seguendo diverse strategie. Gli obiettivi da raggiungere sono:

- Bilanciamento del carico tra le diverse zone
- Minimizzazione della superficie di interfaccia tra le zone

Poiché, per procedere con il calcolo, i processi hanno necessità di sincronizzare i dati in punti precisi del codice, e in ogni caso più volte durante un'iterazione, il primo obiettivo tende ad evitare situazioni in cui tutti i processi si fermino in attesa di dati provenienti da uno soltanto che ha maggior lavoro da compiere, come schematizzato in figura 2.8. Il secondo obiettivo è invece legato alla ricerca della minimizzazione della quantità di dati da scambiare ad ogni sincronizzazione, valore strettamente legato con la dimensione dell'interfaccia tra i vari domini.

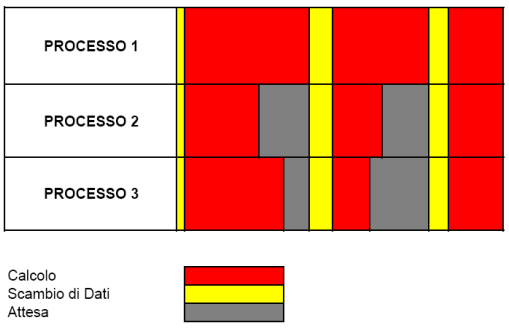


Figura 2.8: Problema di bilanciamento.

Tra i vari algoritmi esistenti è stato scelto `METIS`, che garantisce un ottimo compromesso tra velocità e qualità del partizionamento. Può inoltre essere eseguito in parallelo, nella versione chiamata `ParMETIS`, permettendo così un bilanciamento del carico in corso di simulazione, qualora i cambiamenti topologici della mesh di calcolo compromettano l'efficienza della soluzione. È inoltre possibile specificare un peso differente per ogni cella del dominio da partizionare, migliorando così il ripartimento del carico in situazioni come la combustione, in cui poche celle sono interessate da un evento particolarmente dispendioso in termini di tempo di calcolo.

Alla fine della fase di partizionamento, ogni processo viene a conoscere le dimensioni globali della mesh e due funzioni, una diretta ed una inversa, di corrispondenza tra ogni nodo nella numerazione locale e il suo omologo nella numerazione globale[9].

Fase di calcolo implicita ed esplicita

L'uso delle librerie `PSBLAS` ha di fatto evitato ulteriori significative modifiche in questa parte del codice. L'unica accortezza è l'introduzione di sincronizzazioni di dati tra i diversi processi di calcolo in opportuni punti del codice.

[9]Risulta evidente quindi come la memorizzazione interna della mesh risulti completamente differente rispetto a quanto implementato nella versione originale di Kiva–3V

Per alcune operazioni, infatti, il metodo di risoluzione deve conoscere i valori delle variabili termodinamiche nelle celle adiacenti. Per celle interne al dominio, nulla cambia rispetto al caso seriale. Per celle adiacenti all'interfaccia tra due processi, invece, è necessario che ognuno di essi conosca un numero sufficiente di *strati* di celle all'interno del dominio vicino. Non sarà possibile cambiare i valori della variabili in tale zona, detta *alone*, ma quelle grandezze saranno indispensabili per il corretto calcolo delle variabili nelle celle interne al dominio. È necessario assicurarsi che la zona di alone venga aggiornata ogni volta che sia necessario. Sono state quindi scritte delle apposite routine per facilitare lo scambio delle grandezze desiderate all'interfaccia, per un numero variabile di strati, sia per le celle che per i nodi di calcolo.

Deformazione e cambiamento topologico della griglia

L'algoritmo iterativo semi-esplicito di *rezoning* esistente in Kiva–3V prevedeva che ogni nodo venisse spostato conoscendo le posizioni dei 6 nodi vicini[10]. Le coordinate prese in esame erano quelle modificate dall'algoritmo stesso, non quelle ad inizio deformazione, creando quindi dei potenziali errori all'interfaccia tra due processi, in caso di esecuzione parallela, a meno

[10]Si ricorda che la griglia di calcolo è formata da soli elementi esaedrici ed è strutturata, anche se multi–blocco.

di sincronizzare le coordinate di tutti i nodi ogni volta che uno solo venisse spostato.

Questa soluzione è ben lontana da essere efficiente, e si è quindi scelto di rendere l'algoritmo completamente esplicito, utilizzando le coordinate iniziali dei nodi per valutare le nuove. E' sufficiente quindi una sincronizzazione ad ogni iterazione dell'algoritmo, anziché ad ogni nodo spostato, ottenendo un corretto comportamento in parallelo e, al contempo, un'ottima efficienza, a scapito della copia dei vettori iniziali delle coordinate.

Il trattamento delle modifiche topologiche della griglia, invece, è stato estremamente più complesso ed ha richiesto molte accortezze particolari per il trattamento di eventi di *snap*, sia per il pistone che per le valvole, in corrispondenza di interfacce tra due o più domini di calcolo.

2.2.3 Valutazione delle prestazioni

Premettendo che la qualità della soluzione parallela è del tutto paragonabile a quella seriale, alcuni test di prestazioni sono stati effettuati su diversi cluster e con diverse interconnessioni di rete. Il motore Renault utilizzato come benchmark per i nuovi solutori introdotti è stato preso in considerazione an-

che per la valutazione dell'efficienza in parallelo[11]. La grandezza misurata è detta *speedup* ed è definita come:

$$S(W,p) = \frac{T_s(W)}{T_p(W,p)} \tag{2.1}$$

dove $T_s(W)$ è il tempo di esecuzione del miglior algoritmo seriale, $T_p(W,p)$ il tempo di esecuzione del programma parallelo considerato, W il problema in esame e p il numero di processori. I casi normalmente possibili sono riportati schematicamente in figura 2.9.

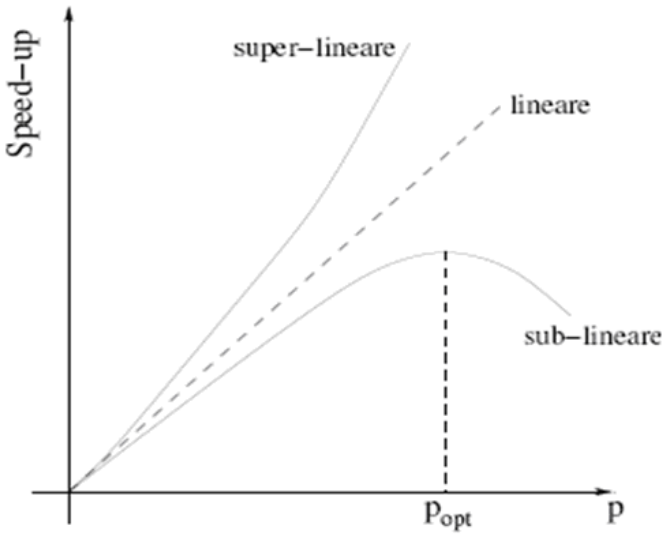


Figura 2.9: Speedup possibili.

Casi superlineari sono prevalentemente legati all'architettura dei moder-

[11]Il cluster di test era composto da processori Intel Pentium 3.06Ghz, 2Gb RAM DDR, rete Myrinet

ni processori il cui accesso alla memoria passa attraverso una *cache* molto veloce ma di dimensioni limitate. Il caso lineare è ciò a cui idealmente si vorrebbe tendere in applicazioni CFD. Il comportamento sub-lineare, con effetti di saturazione quando il tempo necessario per gli scambi di dati diventa prevalente rispetto al tempo di calcolo vero e proprio è invece il caso più comune.

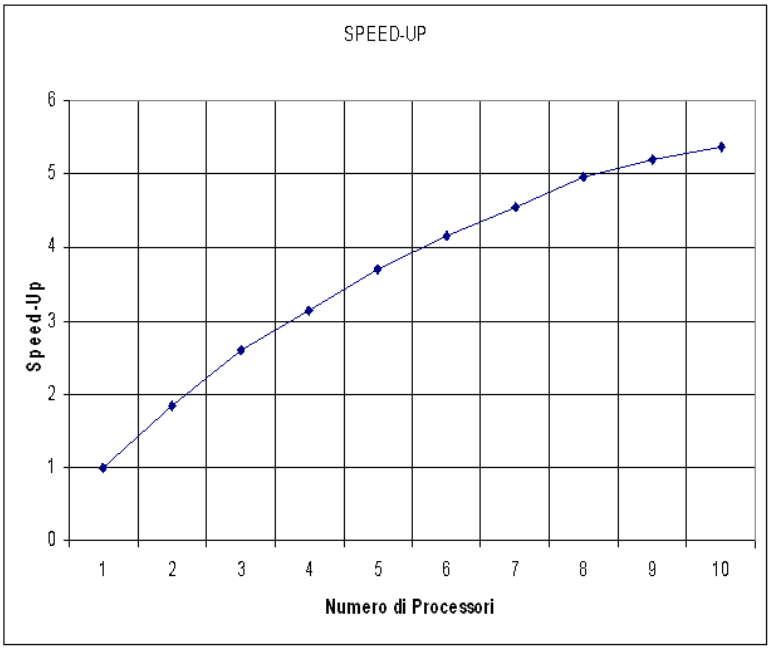


Figura 2.10: Speedup per test case Renault.

Il risultati ottenuti per il caso test Renault sono riportati in figura 2.10. Per evidenziare l'importanza della rete di comunicazione, sono stati effettuati alcune prove utilizzando un cluster di computer messo a disposizione dal CA-

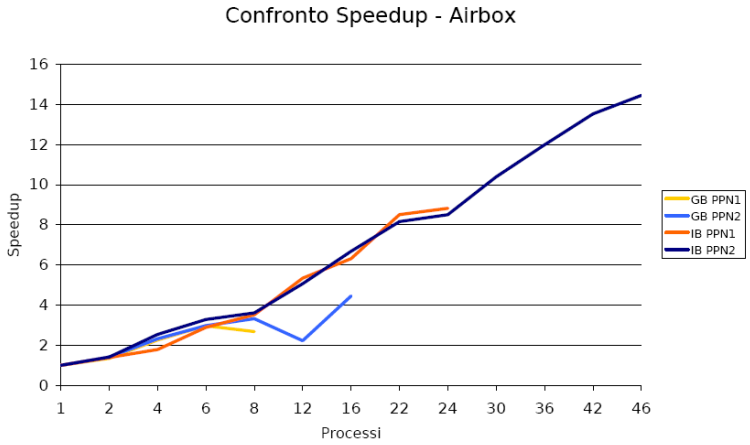


Figura 2.11: Speedup per test case senza mesh mobili.

SPUR[12] di Roma. Il cluster è formato da nodi biprocessore AMD Opteron™ 250 con 4Gb di RAM e 1Mb di memoria cache l'uno. Le reti di comunicazione utilizzate sono state una normale Gigabit ed una InfiniBand Silverstorm InfiniHost III Ex HCA[13]. In figura 2.11 sono riportati i risultati di un test case senza superfici in movimento, la cui griglia di calcolo è costituita da 390228 celle e 440109 vertici. Il grafico mette in risalto l'importanza della rete di comunicazione utilizzata. Importanza ancora più evidente nel caso in cui siano presente mesh mobili, dove gli algoritmi di rezoning richiedono ingenti scambi di dati tra i processi di calcolo, come mostrato in figura 2.12,

[12]Consorzio interuniversitario per le Applicazioni di Supercalcolo Per Università e Ricerca

[13]Questa rete ha una latenza di circa 5 μs ed una banda costante misurata di 960 Mb/s

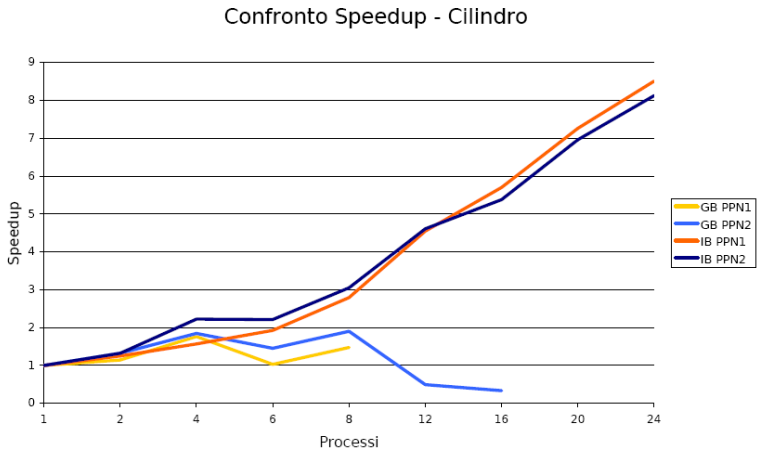


Figura 2.12: Speedup per test case con mesh mobili.

in cui il caso in esame è costituito al massimo da 185995 celle e 202488 nodi.

Un'ulteriore valutazione di efficienza del codice parallelo è stata effettuata per mezzo di una simulazione in prossimità del punto morto superiore di combustione, con reazioni chimiche attivate. La mesh è la stessa utilizzata per la prova con mesh mobili sopra riportata ed una rappresentazione della distibuzione della frazione di gas combusti nel cilindro durante la combustione viene riportata in figura 2.13. La simulazione è stata effettuata per un fissato intervallo angolare attorno al punto di accensione, a partire da una soluzione intermedia, corrispondente a circa 90 time–step.

In tabella 2.1 vengono riportati i dati di tempo e speedup, complessivi e per time–step, della simulazione effettuata su diverse configurazione del

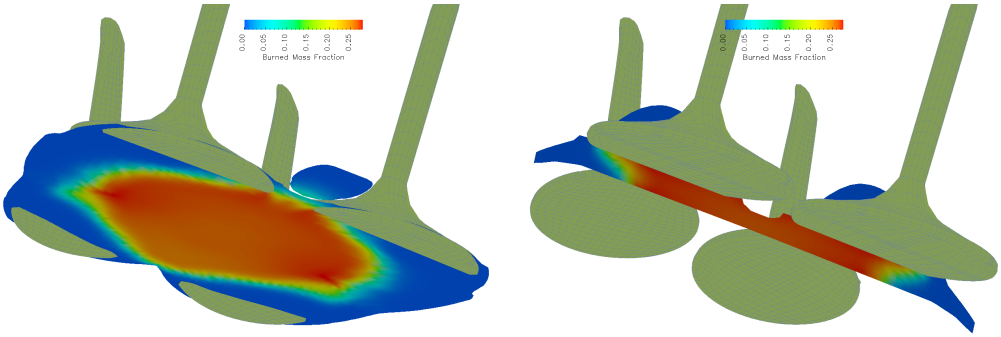


Figura 2.13: Distribuzione della frazione di gas combusti

cluster in uso[14]. La scalabiltà dell'applicazione è buona, anche su un elevato numero di processi, nonostante la griglia di calcolo sia abbastanza piccola. Questo è dovuto da un lato all'aumentato carico per cella dovuto alla combustione, cosa che però può portare ad uno sbilanciamento del carico, qualora la distribuzione di celle che sono interessate da reazioni chimiche non sia uniforme tra i vari processi, dall'altro all'elevata efficienza della rete di comunicazione utilizzata che permette di ridurre sensibilmente il tempo impiegato per lo scambio di dati, se confrontata con una più economica interconnessione Gigabit. Anche con 42 processori, generalmente utilizzati per griglie di calcolo molto più grandi e dettagliate di quella in esame, l'efficienza si mantiene sopra il 50%. È da notare come l'efficienza e il valore di speedup assoluti diminuiscano a 6 e 12 processi a causa dell'aumento del numero di

[14]Si tratta, anche in questo caso, del cluster Opteron messo a disposizione dal CASPUR.

# Processi	Tempo [m]	Speedup	Cicli	[m]/ciclo	Speedup/ciclo
1	114.04	1.00	90	1.27	1.00
2	66.64	1.71	92	0.72	1.75
4	49.87	2.29	96	0.52	2.44
6	44.83	2.54	121	0.37	3.42
8	22.28	5.12	90	0.25	5.12
12	18.36	6.21	103	0.18	7.11
16	12.17	9.37	90	0.14	9.37
20	9.94	11.47	90	0.11	11.47
24	8.73	13.06	90	0.10	13.06
30	6.02	18.94	90	0.07	18.94
36	6.18	18.45	90	0.07	18.45
42	5.23	21.80	90	0.06	21.80

Tabella 2.1: Tempo di calcolo per test case con combustione

time–step necessario per coprire lo stesso intervallo angolare. Questa è una conseguenza degli effetti del partizionamento del dominio sulla convergenza del solutore lineare.

Capitolo 3

Nuove caratteristiche

Vengono qua di seguito descritte le più importanti novità introdotte nel codice. Alcune delle modifiche effettuate sono strettamente legate alla nuova struttura del programma, sia per quanto riguarda la parallelizzazione sia sotto il profilo della gestione delle strutture dei dati. Altre funzionalità sono invece state sviluppate per far fronte ad esigenze relative ai diversi problemi che sono stati affrontati.

3.1 File di ingresso modulare

La struttura del file di ingresso di Kiva, `itape5`, è stata completamente modificata. Originariamente si trattava infatti di una lista ordinata di parametri di ingresso, molto rigida e difficilmente espandibile. Si è provveduto quindi ad

implementare una struttura modulare a blocchi, con una definizione di incompatibilità e dipendenze. È così possibile aggiungere sezioni di configurazione per caratteristiche nuove senza compromettere il corretto funzionamento delle altre componenti.

3.2 Condizioni al contorno

Il trattamento delle condizioni al contorno è stato interamente riscritto. La limitazione presente in KIVA–3V di sole quattro possibili condizioni al contorno differenti è stata rimossa. Il programma è ora in grado di gestire contemporaneamente un numero virtualmente illimitato di *boundary* diversi, o dello stesso tipo ma con diversi valori, costanti o variabili. È stata inoltre implementata uno speciale tipo di valvola di boundary, ovvero senza il cilindro né alcuna zona della griglia presente dopo di essa.

Questa nuova funzionalità, come si può facilmente immaginare, estende di molto i campi di applicazione del programma. È stata inoltre la base indispensabile per lo sviluppo dell'accoppiamento con il codice 1D descritto in seguito. Allo stato attuale è possibile gestire un numero qualsiasi di parti 3D, collegate direttamente all'ambiente esterno o collegate tra loro da una struttura modellata con il codice 1D comunque complessa. È ovvio come le condizioni al contorno da gestire diventino facilmente ben più di quattro.

3.3 Mesh mobili

3.3.1 Cambiamenti nella numerazione locale dei nodi

Originariamente, le strutture dei dati che gestivano la griglia in KIVA–3V erano dimensionate sul massimo numero di nodi possibile e tutti i nodi spenti venivano memorizzati in queste strutture nelle prime $ifirst - 1$ posizioni, essendo $ifirst$ l'indice della prima cella (o nodo) attiva. Ora, invece, ogni processo ha la propria numerazione locale, relativa alla parte di volume ad esso assegnata, e può utilizzare due funzioni per trovare l'indice globale di un nodo locale e viceversa, ma i nodi *spenti* non vengono più conservati. Lo spazio degli indici precedente a *ifirst* contiene solamente delle celle fittizie necessarie per calcolare i flussi attraverso i boundary. Un evento di snap che rimuove un piano di celle elimina completamente i nodi cancellati, mentre uno che aggiunge un piano di celle ricostruisce le celle necessarie, tenendo naturalmente in conto il partizionamento del dominio.

Nel momento in cui si devono aggiungere o togliere piani di celle, la scelta più semplice da seguire era l'assegnazione dell'intero compito ad uno solo dei processi, in numerazione globale, e quindi eseguire un nuovo partizionamento del dominio. Non è però questa, ovviamente, la soluzione più vantaggiosa, né in termini di tempo né di memoria richiesta.

Si è scelto quindi di far sì che ciascun processo detentore di un insieme

di nodi sulla superficie topologica del pistone, che non deve ora necessariamente essere piatta, come verrà illustrato più avanti, aggiunga nodi alla sua numerazione locale o unisca due celle, a seconda della direzione del moto. Al momento di rimuovere piani di celle, può spesso capitare che le celle unite non appartengano allo stesso processo, cioè giacciano sul confine tra due partizioni. Questo tipo di eventi deve essere opportunamente gestito, contrassegnando le celle in esame al fine di trattarle al di fuori della routine di snap standard, così da permettere ai due processi coinvolti di scambiarsi tutte le informazioni necessarie.

3.3.2 Generalizzazione del volume nel cilindro

Nella versione originale di Kiva–3V, il pistone veniva considerato topologicamente piatto, mentre una eventuale *bowl* al di sotto della superficie veniva trattata separatamente. Allo stato attuale di sviluppo, invece, la distinzione tra bowl e normale zona fluida interna al cilindro è stata interamente eliminata, mentre la superficie del pistone può assumere un qualsiasi profilo topologico, a seconda di ciò che la geometria richiede. Quindi le valvole possono muoversi attraverso tutta la zona di *squish*, anche al di sotto della superficie topologica dell'originale pistone piatto. L'unica limitazione è la presenza di almeno due celle fluide tra due qualsiasi superfici solide, per poter corret-

tamente risolvere le equazioni della fluidodinamica. Come si può facilmente comprendere, questo permette di trattare geometrie molto più complesse di quanto non si possa fare con la versione originale del programma. Si pensi, a titolo d'esempio, ad un motore per applicazioni sportive, in cui la testa del pistone sia sagomata in maniera tale da accogliere la valvola, come mostrato in figura 3.1.

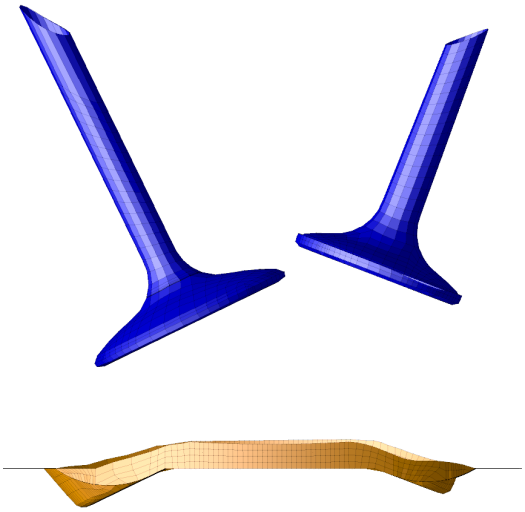


Figura 3.1: Pistone con superficie fortemente sagomata

3.3.3 Miglioramento dello *snap* valvola

Durante lo spostamento delle valvole, lo stelo non veniva mosso, mentre solo la testa della valvola era interessata da spostamenti ed eventi di snap. Lo "spigolo topologico" tra la superficie superiore della testa della valvola e

lo stelo veniva quindi solitamente posizionato subito prima del cambio di curvatura della superficie, come mostrato in figura 3.2.

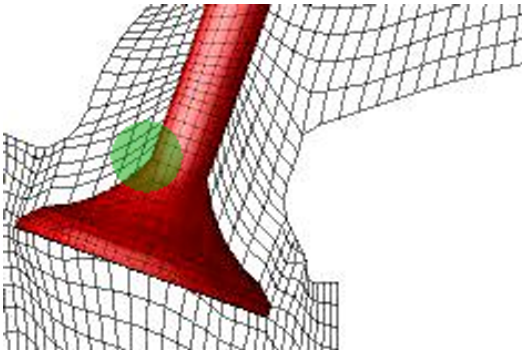
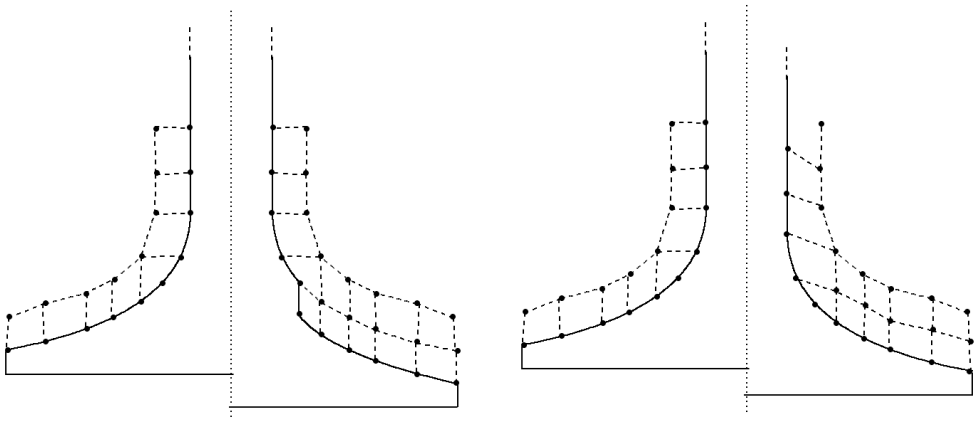


Figura 3.2: Spigolo topologico tra stelo e testa della valvola

Quando questa scelta non era possibile, però, a causa della conformazione della geometria in esame, gli eventi di snap potevano indurre una deformazione della forma della valvola, come mostrato in figura 3.3a. È stata quindi implementata una nuova strategia per il movimento delle valvole, trattandole interamente, compreso lo stelo, come un oggetto rigido ed inserendo o rimuovendo piani di nodi dalla parte superiore dello stelo stesso, invece che dalla parte superiore della testa. Il risultato di questa nuova metodologia è rappresentato in figura 3.3b.

Osservando l'immagine si può capire come ora sia richiesto un algoritmo di rezoning anche attorno allo stelo valvola.



(a) Snap valvola originale (b) Nuova strategia

Figura 3.3: Valvola deformata dopo un evento di snap originale a confronto con la nuova strategia adottata

3.4 Trattamento di gocce e spray

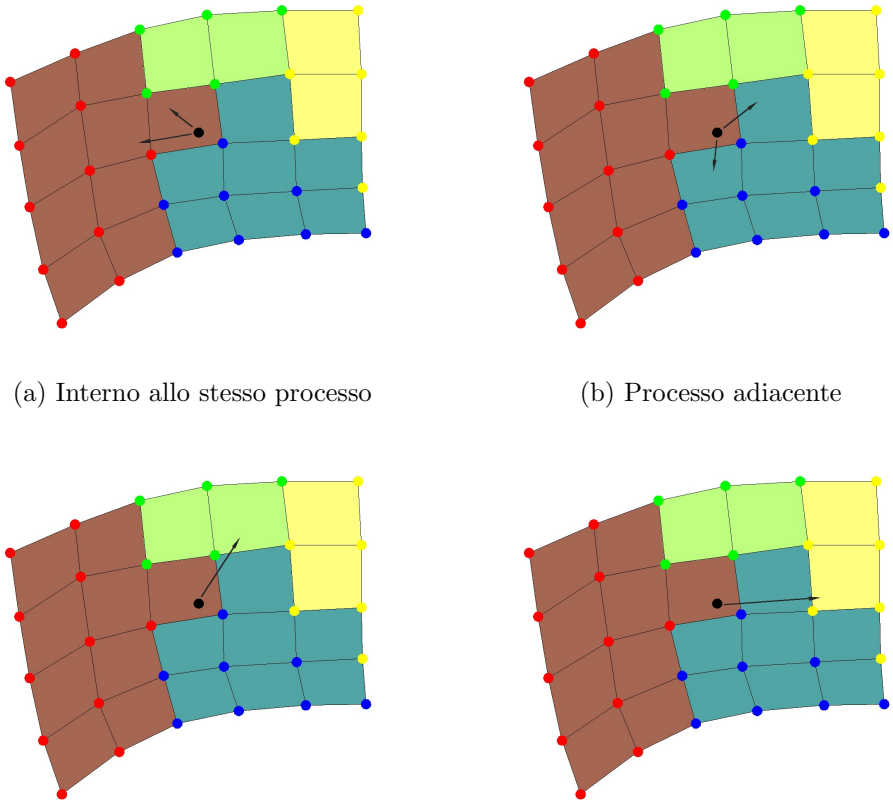
Ogni goccia viene assegnata ad uno dei processi che compongono la simulazione in base alla sua posizione, cioè a seconda del "proprietario" della cella in cui è inclusa. Quando una goccia attraversa il confine tra due partizioni, deve essere rimossa dalla lista di quelle appartenenti al primo processo ed aggiunta alle gocce appartenenti al secondo. Sono state scritte, quindi, alcune routine che permettono la spedizione e la ricezione di gocce tra processi. Viene gestito anche il caso poco frequente in cui una particella attraversi due confini tra partizioni in un solo intervallo di tempo di simulazione, come mostrato in figura 3.4. Particolare attenzione è stata inoltre prestata per ridurre

il tempo necessario ad individuare la posizione topologica di una goccia sia durante la fase di iniezione, sia dopo ogni time step. Un trattamento particolare, inoltre, è stato necessario per il film presente sulle superfici nel caso in cui debba cambiare processo di appartenenza a seguito di un evento di snap o di un nuovo partizionamento, soprattutto quando il film viene a trovarsi su superfici mobili.

3.5 Rezoning della griglia

L'algoritmo di rezoning è stato reso completamente esplicito per esigenze di parallelizzazione, ovvero, durante il ciclo sui nodi di griglia per trovare la nuova posizione, vengono aggiornate le coordinate del solo nodo in esame, tenendo presente le posizioni dei nodi vicini e delle distanze topologiche da superfici fisse o mobili nelle sei direzioni. L'algoritmo converge quando ogni nodo del mesh viene spostato meno di un fissato valore durante un'iterazione.

A causa delle complesse geometrie dei motori, è praticamente impossibile individuare un insieme di pesi per le medie tali da garantire una buona qualità della griglia per tutti i nodi della mesh, indipendentemente dalle posizioni delle superfici mobili. È quindi necessario definire diverse porzioni della griglia logica, potenzialmente variabili con l'angolo di manovella, e muovere i nodi in esse inclusi con differenti valori dei pesi.



(a) Interno allo stesso processo

(b) Processo adiacente

(c) Fuori dall'alone

(d) Due strati

Figura 3.4: Possibili spostamenti di una goccia: internamente al processo che la detiene (a), verso un processo adiacente all'interno dell'alone (b), verso un processo adiacente ma fuori dall'alone (c), verso un processo non confinante (d)

Per evitare di ricompilare il codice al variare di un peso, la definizione di una zona o qualsiasi dato necessario all'algoritmo di rezoning, tutte le informazioni ad esso necessarie possono essere specificate nel file di input. Questa

caratteristica è stata sviluppata con l'obiettivo di ottenere una struttura *user–friendly* per i dati di ingresso ed allo stesso tempo ottenere prestazioni, in termini di tempo di calcolo, simili a quelle della routine compilata. Questo è stato possibile grazie all'utilizzo di alcune caratteristiche dello standard Fortran95, come tipi di dati derivati, variabili allocabili e puntatori.

Vengono di seguito descritti l'uso ed il funzionamento, insieme alle strutture dati utilizzate, del blocco di `Itape5` in cui è possibile definire sia i blocchi di vertici a cui assegnare un particolare valore di `Zsquish`, sia i pesi da utilizzare in `Rezpent` per calcolarne la posizione.

3.5.1 Utilizzo

Il blocco del file di ingresso comincia con la riga

```
#0051 ********** ASSEGNAZIONE ZSQUISH E PESI PER REZPENT ***********
```

e termina con l'inizio del blocco successivo (o con la fine dell'`Itape5`).

All'interno di esso, ogni riga che ha '`!`' come primo carattere viene considerata riga di commento ed ignorata. Ogni comando, invece, termina con un '`;`'. È così possibile scrivere più comandi su una sola riga o un comando su più righe.

Le parti fondamentali di cui è composto il blocco sono 3, nell'ordine:

- Dichiarazione di alcuni parametri comuni;

- Definizione dei valori di `Zsquish`
- Definizione dei pesi per `Rezpent`

Blocchetto common

La dichiarazione dei parametri comuni è fatta in un blocco del tipo

```
common {
  ! Parametri com numero di iterazioni e tolleranza
  iter  =1000;
  tol   =0.04;
  kglobx=1.;
  kgloby=1.;
  kglobz=1.;
}
```

in cui '`iter`' indica il numero massimo di iterazioni ammissibili in `Rezpent`, '`tol`' la tolleranza di convergenza dell'algoritmo mentre '`kglobx`', '`kgloby`' e '`kglobz`' sono i fattori moltiplicativi dei pesi proporzionali alle distanze topologiche da pareti fisse secondo le tre direzioni.

Sezione Zsquish

La sezione in cui viene specificato come assegnare i valori di `Zsquish` inizia con la stringa

```
[zsquish]
```

a cui seguono un numero arbitrario di *regole* che Kiva interpreterà. Queste sono suddividibili in due categorie, descritte di seguito.

Definizione tramite gli estremi di un *parallelepipedo* Viene assegnato un assegnato valore di zsquish ad una regione topologica delimitata da un intervallo di valori per le direzioni `x`, `y` e `z`. Ad esempio:

```
zsquish=7 {
  ! ZONA TRA LE VALVOLE DI ASPIRAZIONE
  reg=1;
  x1=14;      x2=19;
  y1=23;      y2=32;
  z1=zmax-10; z2=zmax-2;
}
```

In particolare, il blocchetto precedente fa sì che venga assegnato un valore di `zsquish` pari a 7 ai nodi della regione 1 (`reg=1;`) che abbiano coordinata topologica `x` compresa tra 14 e 19, `y` compresa tra 23 e 32 e `z` compresa tra il massimo valore assunto da `z` nella regione 1 meno 10 e lo stesso valore meno 2.

In generale, per ogni coordinata è possibile specificare:

- Un valore costante (‘`x1=14;`’)
- Un valore relativo al minimo valore di una coordinata all’interno della regione specificata(‘`y1=ymin;`’ o ‘`x2=3+xmin;`’ o ‘`y2=ymin+4;`’)
- Un valore relativo al massimo valore di una coordinata all’interno della regione specificata (‘`y1=ymax;`’ o ‘`x2=-3+xmax;`’ o ‘`y2=ymax-4;`’)

Definizione tramite vicinanza Alternativamente, è possibile identificare i nodi a cui assegnare un determinato valore di zsquish per mezzo della posizione rispetto ai nodi con un assegnato valore di `idface`, di `fv` o di `zsquish`, anche se quest'ultima possibilità è da usarsi con particolare cautela.

Un esempio di definizione tramite vicinanza è il seguente:

```
zsquish=4 {
  reg=1;
  ! Piano sotto una valvola spostato a dx
  idface(kp(kp(im(im))))=1;
}
```

Il blocchetto precedente fa sì che venga assegnato un valore di `zsquish` pari a 4 ai nodi della regione 1 che abbiano il vicino in direzione `im,im,kp,kp` con valore di idface pari ad 1. Il risultato è un insieme di nodi posto due piani sotto la valvola identificata da idface 1 e spostata a destra (topologicamente) di due.

Per motivi di immagazzinamento di dati, il numero massimo di ricorsioni nell'identificare un vicino è pari a 5.

Sezione pesi Rezpent

La sezione in cui viene specificato come assegnare i valori di `Zsquish` inizia con la stringa

```
[rezpent]
```

a cui seguono una *regola* di default (ovvero per i nodi aventi zsquish pari a 1) più un numero arbitrario di *regole* che Kiva interpreterà. È inoltre possibile specificare i pesi per il movimento dei nodi attorno agli steli valvola con `irez 5`, in particolare un comportamento di default più un numero arbitrario di particolarizzazioni per ogni valvola.

Pesi per nodi con zsquish assegnato La sintassi da utilizzare per i nodi aventi zsquish 1 o diverso da 1 è praticamente la stessa. L'unica differenza sta nel fatto che il blocchetto che identifica zsquish=1 è identificato con la label `default` e deve essere dichiarato prima di qualsiasi altro. Due esempi sono i seguenti:

```
default {
x = jm*1+jm*gym*1 + jp*1+jp*gyp*1 + km*1+km*gzm*1 + kp*1+kp*gzp*1+
    im*1+im*gxp*gym*0.2+im*gxp*gyp*0.2 + ip*1+ip*gxm*gym*0.2+
    ip*gxm*gyp*0.2;
y = im*1+im*gxm*1 + ip*1+ip*gxp*1 + km*1+km*gzm*1 + kp*1+kp*gzp*1+
    jm*1+jm*gyp*gxm*0.2+jm*gyp*gxp*0.2 + jp*1+jp*gym*gxm*0.2+
    jp*gym*gxp*0.2;
z = jm*1+jm*gym*1 + jp*1+jp*gyp*1 + im*1+im*gxm*1 + ip*1+ip*gxp*1+
    km*0.95 + kp*1.05;
}
```

```
zsquish=3{
crank=120.5 : 650.;
x = jm*1+jm*gym*1 + jp*1+jp*gyp*1 + km*1+km*gzm*1 + kp*1+kp*gzp*3+
    im*1+im*gxp*gym*0.2+im*gxp*gyp*0.2 + ip*1+ip*gxm*gym*0.2+
    ip*gxm*gyp*0.2;
y = y(default);
z = jm*1+jm*gym*1 + jp*1+jp*gyp*1 + im*0.3+im*gxm*1 + ip*0.3+
    ip*gxp*1+km*5 + kp*5;
}
```

I due blocchetti precedenti assegnano i pesi da utilizzare in `Rezpent` per i nodi aventi zsquish 1 e 3, rispettivamente.

Ogni comando è composto da una *somma*[1] di diversi addendi, che possono essere generalmente suddivisi in due tipologie:

- Pesi non dipendenti da distanze verso pareti (nodi non spostabili da `Rezpent`);
- Pesi dipendenti da distanze verso pareti (nodi non spostabili da `Rezpent`).

Alla prima categoria appartengono gli elementi tipo `jm*1`[2] o `kp*1.05`. Sono elementi costituiti da due parti, prima e dopo l'asterisco. La prima parte è la direzione topologica (`im`,`ip`,`jm`,`jp`,`km`,`kp`), la seconda è il valore del peso.[3]

Alla seconda categoria appartengono gli elementi tipo `ip*gxp*1` oppure `jp*gym*gxm*0.2`. Sono elementi costituiti da tre parti, separate tra loro dal primo e dall'ultimo asterisco. Infatti vengono interpretati come:

```
ip*gxp*1        → ip    gxp      1
jp*gym*gxm*0.2 → jp  gym*gxm  0.2
```

[1] Non si tratta di una vera operazione di somma, ma di una media pesata.

[2] Questo elemento è superfluo, perché il valore 1 è il default per questo genere di pesi.

[3] Nelle precedenti versioni di `rezpent`, `x=jm*1.2+jm*gym*1+ ...` corrisponde a `xn=((1.2*kloc+gym ...)*x(jm)+ ...)/(1.2+gym+ ...)`

Si veda la nota 3 per un esempio di interpretazione di questi elementi e il paragrafo 3.5.2 per informazioni sull'immagazzinamento, spedizione ed utilizzo dei valori letti.

Per ultimo, è opportuno notare due particolarità del blocchetto che assegna i pesi di rezpent per i nodi aventi zsquish 3 (il secondo di quelli sopra riportati).

Prima di tutto, è specificato un'intervallo di angolo di manovella in cui valgono i pesi specificati ('`crank=120.5 : 650.;`')[4]. Dopo la lettura completa del blocco di `itape5` in esame viene effettuato un controllo sull'eventuale sovrapposizione degli intervalli di angoli e sulla congruenza tra definizioni di zsquish e di pesi di rezpent. In particolare, non è ammesso un valore dell'angolo per il quale sia definito un certo valore di zsquish ma non i rispettivi pesi di rezpent. Il contrario è invece consentito.

L'assegnazione dei pesi per il calcolo della coordinata `y` viene effettuata facendo riferimento ad un altro valore di zsquish. Vi è infatti il comando `y = y(default);`, del tutto equivalente a `y = y(1);`. È possibile, però, fare riferimento a qualunque zsquish, purché già assegnato, purché non vi sia ambiguità, ad esempio per valori diversi dei pesi al variare del crank, e pur-

[4]Questo stesso comando può essere usato all'interno dei blocchetti di definizione dei nodi a cui assegnare un particolare valore di `zsquish`

ché venga fatto riferimento alla stessa coordinata. Non è cioè ammesso un comando tipo `y=x(3);`, ma lo è `x=x(3);`

Pesi per nodi attorno agli steli valvola Nel caso in cui venga utilizzato `irez=5` è necessario definire almeno un comportamento di default per i nodi attorno agli steli valvola. In aggiunta, è possibile definire dei comportamenti specifici per i nodi da ciascun lato (topologico) di ciascuna valvola.

Uno schema della valvola[5], con la nomenclatura utilizzata, viene riportato in figura 3.5:

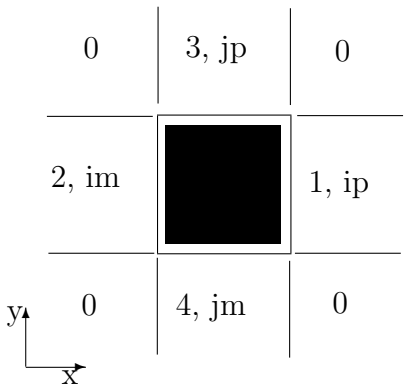


Figura 3.5: Zone attorno allo stelo valvola per l'algoritmo di rezoning

I pesi influenzano due strati di celle attorno allo stelo valvola e possono essere assegnati *radialmente* o identificando le diverse zone singolarmente. Ad esempio si può avere:

[5]Per ora viene supposto che lo stelo della valvola sia orientato come z.

```
def_valve {
rx = 1.3 0.7 0 0 ;
ry = 1.3 0.7 0 0 ;
rz = 4   1   1 1 ;
}
valve=1 {
1x = 0.5 2.5 0 0;
0yip = 2;
}
valve=2 {
rx = 2.5 0.5 0 0 ;
ry = 2.3 0.7 0 0 ;
rz = 4   1   1 1 ;
}
```

Negli esempi precedenti, i pesi di default vengono assegnati radialmente (i 4 numeri indicano il peso lato valvola, lato opposto e gli altri due, rispettivamente), per la valvola 2 vengono sovrascritti tutti, ancora una volta radialmente, mentre per la valvola 1 vengono assegnati i pesi per la coordinata x nella zona 1 (nell'ordine `ip`, `im`, `jp`, `jm` rispettivamente) e il peso della coordinata y nella direzione `ip` negli *angoli* della valvola, mentre tutti i valori non assegnati vengono assegnati automaticamente uguali a quelli di default.

Sintassi:

Assegnazione radiale Come visto sopra, la sintassi è del tipo:

```
r{x|y|z} = # # # #;
```

dove '`r`' indica assegnazione radiale, mentre '`x`', '`y`' o '`z`' indicano la

coordinata per la cui determinazione si stanno assegnando i pesi. I quattro numeri dopo l'uguale indicano i pesi dei nodi "lato stelo", "lato opposto" e gli altri due (che dovrebbero essere tra loro uguali, per ragioni di simmetria).

Ad esempio, '`rx = 1.3 0.7 0 0 ;`' fa sì che per determinare la coordinata x dei nodi nella zona 4, facendo riferimento alla figura sopra riportata, venga utilizzato un peso pari a 1.3 in direzione `jp`, 0.7 in direzione `jm` e 0 nelle direzioni `ip` e `im`. Nella zona 1, invece, lo stesso comando fa sì che venga utilizzato un peso pari a 1.3 in direzione `im`, 0.7 in direzione `ip` e 0 nelle direzioni `jp` e `jm`.

Assegnazione per zone – tutti i pesi È possibile assegnare, con una sola istruzione, i pesi nelle quattro direzioni per una delle cinque zone in cui sono suddivisi i nodi intorno allo stelo valvola. La sintassi è del tipo:

```
{0|1|2|3|4}{x|y|z} = # # # #;
```

dove il primo numero indica la zona in considerazione, mentre la seconda lettera indica a coordinata per la cui determinazione si stanno assegnando i pesi. I quattro numeri dopo l'uguale indicano i pesi dei nodi nelle direzioni `ip`, `im`, `jp`, `jm` rispettivamente.

Ad esempio '`2y = 0.5 2.5 0.1 0;`' fa sì che per determinare la coor-

dinata y dei nodi nella zona 2 venga utilizzato un peso pari a 0.5 in direzione `ip`, 2.5 in direzione `im`, 0.1 in direzione `jp` e 0 in direzione `jm`.

Assegnazione per zone – singolo peso È anche possibile assegnare o modificare rispetto al valore di default il valore del peso per i nodi in una delle cinque zone attorno alla valvola, per una coordinata e in una sola direzione. La sintassi è del tipo:

```
{0|1|2|3|4}{x|y|z}{[{1|2|3|4}|{ip|im|jp|jm}]} = #;
```

dove il primo numero indica la zona in considerazione, la seconda lettera indica a coordinata per la cui determinazione si sta assegnando il peso, mentre il terzo numero, o le ultime due lettere, indica la direzione topologica del vicino a cui si sta assegnando il peso.

Ad esempio '`0yip = 2;`', equivalente a '`0y1 = 2;`', fa sì che per determinare la coordinata y dei nodi nella zona 0 (le *diagonali*) venga utilizzato un peso pari a 2 in direzione `ip`.

Note conclusive Per poter gestire diversamente le cinque zone attorno agli steli valvola, vengono generate automaticamente le definizioni di 5 valori di zsquish, a seguito dell presenza del blocchetto `def_valve`, più 5 per ogni valvola con particolarizzazioni.

La numerazione di questi valori di zsquish viene effettuata automaticamente a partire dal valore massimo tra quelli definiti in `Itape5`.

3.5.2 Strutture dati

Per la memorizzazione e l'utilizzo delle informazioni lette nel blocco di lettura di `Itape5` sulle definizioni di `zsquish` e di pesi per `rezpent`, è stata creata una struttura dati di tipo derivato, composta a sua volta da altri tipi di dato derivato[6].

Una schematizzazione della struttura dati viene riportata in figura 3.6.

Spedizione dei dati

È stato necessario scrivere una routine di spedizione e una di ricezione delle informazioni lette dal processo `root` agli altri processi del run parallelo[7]. Vengono spedite inizialmente le variabili lette nel blocchetto `common` (vedi 3.5.1) e successivamente tutte le altre, organizzando i dati in due buffer, uno di interi e l'altro di reali. La struttura del buffer viene descritta in dettaglio nell'appendice A. Di seguito si riportano alcune importanti considerazioni sulla loro struttura.

[6]Per la definizione dei tipi, vedere il modulo `mod_zsquish.f90`

[7]Le routine si chiamano `BSendTZsqRez` e `BRecvTZsqRez` e richiedono come argomento una variabile di tipo `TZsqRez`, allocata e riempita per la prima, non allocata per la seconda.

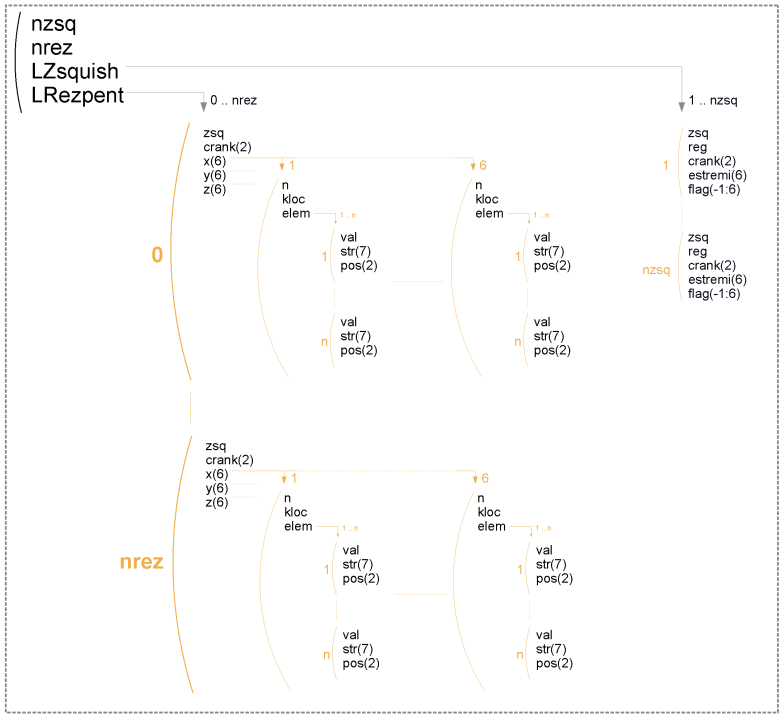


Figura 3.6: Schema della struttura dati utilizzata

Buffer reale Come si evince dalla tabella A.1 a pagina 139, la dimensione del buffer è:

$$2 \cdot nzsq + \sum_{j=0}^{nrez} \left[2 + \sum_{i=1}^{6 \cdot 3} (n_{j,i} + 1) \right] \tag{3.1}$$

I valori interi usati come limiti di alcuni intervalli, come $nrez$, $n_{0,1}$, etc. . . , vengono spediti all'interno del buffer dei valori interi. Il valore complessivo della lunghezza del buffer, tanto di quello reale come di quello intero, viene invece spedito a parte, per consentire una corretta allocazione alla routine ricevente.

Il buffer è suddiviso in due parti, la prima contenente i valori degli intervalli di crank nelle regole che definiscono i valori di `zsquish`, che sono gli unici campi reali di quel tipo di dato, la seconda contenente gli intervalli di crank ed i valori dei pesi di Rezpent.

Buffer intero Come si evince dalla tabella A.2 a pagina 140, la dimensione del buffer è:

$$2 + 18 \cdot nrez + 16 \cdot nzsq + \sum_{j=0}^{nrez} \left[1 + \sum_{i=1}^{6 \cdot 3} (9 \cdot n_{j,i}) \right] \tag{3.2}$$

Come si vede dalla tabella, il buffer è costituito essenzialmente di tre parti. La prima contiene alcuni valori necessari per conoscere il numero di dati da ricevere e le dimensioni su cui allocare i campi non statici della variabile. La seconda contiene invece i valori interi presenti nella parte relativa alle definizioni delle regole per assegnare i corretti `zsquish` ai nodi. La terza ed ultima, infine, contiene le informazioni relative ai pesi per `Rezpent`, compresi i valori `ASCII` dei caratteri componenti le stringhe `str`[8].

Utilizzo dei pesi di rezpent

Nella routine `Rezpent` *tradizionale*, il calcolo della coordinata `x` di un nodo potrebbe essere scritto come:

[8] In fase di runtime, in realtà, non vengono mai usate le stringhe (tipo `gxp` o `gyp*gxp`), ma solo i valori ottenuti dal loro preprocessamento e memorizzati nel vettore `pos`.

```
if (zsquish(i4)==4) &
&xn=(...+(0.7*kloc+1.1*gxp*gyp+0.9*gym)*x(jp) ...)/&
   &((...+0.7+...)*kloc+...+1.1*gxp*gyp+0.9*gym)
```

La corrispondente parte in `Itape5` è

```
zsquish=4 {
x = ... jp*0.7+jp*gxp*gyp*1.1+jp*gym*0.9 ...
y = ...
z = ...
}
```

Il blocchetto sopra riportato viene memorizzato nella variabile `LZsqRez` di tipo `TZsqRez`, la cui struttura è riportata in figura (3.6). Se quello in esame è l'i–esimo blocchetto letto, i dati memorizzati saranno:

```
LZsqRez%LRezpent(i)%x(3)%kloc == 0.7
LZsqRez%LRezpent(i)%x(3)%n    == 2
LZsqRez%LRezpent(i)%x(3)%elem(1)%val == 1.1
LZsqRez%LRezpent(i)%x(3)%elem(1)%str == "gxp*gyp"
LZsqRez%LRezpent(i)%x(3)%elem(1)%pos(1:2) == (/1,3/)
LZsqRez%LRezpent(i)%x(3)%elem(2)%val == 0.9
LZsqRez%LRezpent(i)%x(3)%elem(2)%str == "gym"
LZsqRez%LRezpent(i)%x(3)%elem(2)%pos(1:2) == (/4,0/)
```

Il campo `pos` di ogni elemento `elem` viene riempito a partire dalla stringa `str` dello stesso elemento. In particolare vengono individuati i due fattori del prodotto in `str` (se il fattore è uno solo, `pos(2)` viene impostato pari a 0), e vengono assegnati agli elementi del vettore intero `pos` due interi, da 1 a 6, corrispondenti alla direzione topologica a cui il peso fa riferimento.

All'interno della routine `rezpent_it5.f90`, per ogni nodo, dopo aver

calcolato i pesi proporzionali alle distanze topologiche dalle pareti, viene costruito il vettore

```
gvec(0:6)=(/1.,gxp,gxm,gyp,gym,gzp,gzm/)
```

È importante a questo punto notare che, dato il campo `pos` di un elemento `elem` visto poco sopra, l'elemento del vettore `gvec` indirizzato da `pos(i)` è pari al valore del peso. In particolare, se si considerano i campi sopra riportati ed utilizzando un puntatore intermedio, per semplificare la notazione e velocizzare l'indirizzamento, si ha:

```
p=>LZsqRez%LRezpent(i)%x(3)
gvec(p%elem(1)%pos(1))=gvec(1)=gxp
gvec(p%elem(1)%pos(2))=gvec(3)=gyp
gvec(p%elem(2)%pos(1))=gvec(4)=gym
gvec(p%elem(2)%pos(2))=gvec(0)=1
```

Il preprocessamento delle stringhe `str` così fatto permette di velocizzare notevolmente il calcolo all'interno della nuove `Rezpent`. Infatti per valutare il contributo della parte di istruzione fin'ora considerata, è sufficiente scrivere:

```
p=>LZsqRez%LRezpent(i)%x(3)
xtmp=p%kloc
do i=1,p%n
  xtmp=xtmp+(gvec(p%elem(i)%pos(1))*&
       &gvec(p%elem(i)%pos(2))*p%elem(i)%val)
end do
```

Nell'implementazione all'interno di `rezpent_it5.f90`, per motivi di efficienza, sono stati utilizzati dei `forall` annidati fra loro. Ad esempio, il calcolo della coordinata `x` di un nodo viene fatta come:

```
ploc=>LZsqRez%LRezpent(jj)
...
! Assegnazione di xn
forall (j=nip:nkm)
  resv1(j,0)=ploc%x(j)%kloc
  forall (i=1:ploc%x(j)%n) resv1(j,i) = &
   &gvec(ploc%x(j)%elem(i)%pos(1))*&
   &gvec(ploc%x(j)%elem(i)%pos(2))*ploc%x(j)%elem(i)%val
  denv(j)=sum(resv1(j,0:ploc%x(j)%n))
  numv(j) = denv(j)*x(vicini(j))
end forall
xn=sum(numv)/sum(denv)
```

Funzionamento di Distsolid

La versione di `distsolid` che utilizza il tipo di dato `TZsqRez`, il cui schema è riportato in figura 3.6, è chiamata `distsolid_it5.f90`. Il tipo di dato che compone la lista `LZsquish` è definito come segue:

```
  type, public  :: TZsquish
    integer          :: zsq = 0
    integer          :: reg = 0
    real(kind(1.d0)) :: crank(2)   = (/0.,720./)
    integer          :: estremi(LenEstremi) =     (/0,0,0,0,0,0/)
    integer(1)       :: flag(-1:LenEstremi) = (/0,0,0,0,0,0,0,0/)
  end type TZsquish
```

Facendo riferimento al seguente blocchetto in `Itape5`

```
zsquish=7 {
  ! ZONA TRA LE VALVOLE DI ASPIRAZIONE
  reg=1;
  crank= 100: 370.5;
  x1=14;      x2=19;
  y1=23;      y2=32;
  z1=zmax-10; z2=zmax-2;
}
```

il significato dei vari campi è il seguente:

zsq Valore dello `zsquish` da assegnare ai nodi che rientrano nella regola corrente (7 nel caso in esame).

reg Regione a cui devono appartenere i nodi (1 nel caso in esame).

crank Estremi dell'intervallo di crank in cui la regola è valida ((/100., 370.5/) nel caso in esame).

estremi Se l'assegnazione degli zsquish è fatta assegnando gli estremi di un parallelepipedo, i sei numeri contengono la parte numerica del valore scritto nel rispettivo blocchetto di itape5, nell'ordine x1,x2,y1,y2,z1,z2. Cioè, nell'esempio corrente $(/14, 19, 23, 32, -10, -2/)$.

Se l'assegnazione degli zsquish viene fatta individuando i nodi per mezzo dei nodi vicini, `estremi(1)` assume il valore della proprietà richiesta (idface, zsquish o fv, a seconda del valore di `flag(1)`), gli altri 5 individuano un percorso per trovare il vicino richiesto.

Se il blocchetto in esame è tra quelli generati automaticamente per individuare i nodi attorno alle valvole, `estremi(1)` contiene il numero della valvola in esame (`idface/2`), o 0 per i valori di default (tutte le valvole senza particolarizzazioni), mentre `estremi(2)` contiene il lato

dello stelo valvola in cui si trova il blocco di nodi da individuare (vedere figura relativa alla topologia attorno alla valvola).

flag Gli elementi in posizione -1 e 0 vengono utilizzati per segnare l'assegnazione o meno della regione e dell'intervallo di crank, rispettivamente, per controllare definizioni duplicate o mancanti. Gli elementi da 1 a 6 assumono significati diversi nel caso in cui l'assegnazione degli zsquish sia fatta attraverso gli estremi di un parallelepipedo o meno.

In particolare, nel caso di parallelepipedo il significato dei flag è, per ogni corrispondente elemento dell'array `estremi`

0 Elemento non inizializzato;

1 Assegnato ad un valore costante;

2 Assegnato rispetto al minimo topologico nella regione;

3 Assegnato rispetto al massimo topologico nella regione;

Nell'esempio in esame, si ha $flag(-1:6)$ $(/1,1,1,1,1,1,3,3/)$.

Nel caso in cui i nodi vengano individuati attraverso la vicinanza ad altri nodi con una certa proprietà, o per zone attorno agli steli valvola, `flag(1)` può valere

4 Zona attorno agli steli valvola (è generato automaticamente);

10 Vicino con `idface` assegnato;

11 Vicino con `zsquish` assegnato;

12 Vicino con `fv` assegnato;

mentre i valori di `flag(2:5)` vengono ignorati.

3.6 Database dei run

Per tenere traccia delle diverse esecuzioni del programma, in maniera da poter facilmente accedere ai dati generati, potenzialmente distribuiti sui dischi locali dei nodi di calcolo, alcune informazioni rilevanti per i run di *produzione* vengono immagazzinate in un database appositamente definito. Per facilitare l'integrazione con Kiva e con i tool esistenti, il database è contenuto in un file di testo e l'interfaccia per accedervi è stata scritta da zero, anziché utilizzate librerie di pubblico dominio disponibili.

È possibile in ogni caso disabilitare l'accesso al database, quando si vogliano ad esempio effettuare delle esecuzioni di prova, utilizzando l'opzione `-local` da linea di comando.

Un esempio di elemento del database, come scritto al suo interno, è il seguente.

Esempio di Record

```
begin (10945 version_1.0)
 idrun = 10945
 nome = * Renault F1 - nuova serie - RPM 16000
```

```
 user =          508
 username = francesco
 stato = Normal termination.
 scratch =  T
 deleted =  T
 idrun_rd =            0
 data_beg = 02.08.2004; 14.50.43
 data_end = 02.08.2004; 14.59.09
 ciclo_beg =           0
 ciclo_end =         100
 crank_beg =   0.000000000000000E+000
 crank_end =    469.661272751870
 time_beg =   0.000000000000000E+000
 time_end =   1.006382578319780E-004
 nprocs =            7
 machines = node1 node2 node3 node5 node6 node7 node9
 pid =  15320  25055  23264   2032   2038  26641  16127
 pathlocale = /home/francesco/job.10945_tmp/
end (10945)
```

Benché, come già scritto, il file del database sia un normale file di testo, la modifica diretta tramite editor di testo è da evitare. Per eseguire le operazioni più comuni sul database è opportuno, invece, utilizzare un programma appositamente sviluppato, chiamato *manage_dump* .

3.6.1 Interfaccia tra Kiva ed il database

Kiva accede al database tramite il modulo `Modlogkiva`, contenente i tipi di dato necessari per leggere ed operare sugli elementi (gestiti come lista record–puntatore) e tutte le routine necessarie per eseguirvi operazioni. A loro volta le routine di alto livello, per eseguire operazioni complesse come

l'estrazione della lista dei nodi da un elemento, o la modifica al momento della fine dell'esecuzione, vengono chiamate esclusivamente all'interno della routine `MakeLog`, per quanto riguarda la creazione o l'aggiornamento di un record, o all'interno della `taperd`, per recuperare alcune informazioni necessarie per la ripartenza, come numero macchine e lista dei nodi dove reperire i dump.

3.6.2 *manage_dump*

manage_dump è il programma da utilizzare per eseguire le operazioni più o meno comuni sia sui dump[9] salvati, sia per recuperare i file dati o grafici, sia per estrarre informazioni o eseguire operazioni di manutenzione sul database dei run.

È prevista sia una modalità interattiva, di default, sia una modalità da linea di comando, dove però, anche per ragioni di sicurezza, non sono implementati tutti i possibili comandi.

Di seguito vengono illustrate le opzioni più comunemente utilizzate. Per le altre, fare riferimento alla guida in linea, ottenibile con `"manage_dump -h"`, oppure `"manage_dump -h -cl"` per le operazioni eseguibili da linea di comando [10].

[9] Con il termine *dump* si intende il file necessario per una successiva ripartenza e salvato da Kiva ad intervalli fissati o in particolari istanti di simulazione, come specificato in input.

[10] Attualmente, per compatibilità con i run effettuati prima dell'introduzione del data-

Per permettere l'utilizzo di diversi database dei run, ad esempio uno per il cluster di calcolo condiviso con tutti gli utenti, uno sul proprio computer locale per memorizzare run di test o di sviluppo, è possibile in qualsiasi momento cambiare database di lavoro all'interno di *manage_dump* .

Quando eseguito in modalità interattiva, il programma si presenta come segue:

```
_____________ manage_dump : menu principale _____________
********** Manage  Dump **********
********** ver.  1.29 **********
********** 31.08.2006 **********
-----------------
 Utente : francesco
-----------------

Scegliere operazione:
---------------------
  1)  Copia/Spostamento da nodi in locale
  2)  Copia/Spostamento da locale a nodi
  3)  Spostamento tra nodi diversi (stesso numero)
  4)  Collezione di n dump in un dump "seriale"
  5)  Divisione di un dump seriale su n processi
  6)  Copia da otape8 a itape7
  7)  Eliminazione di cartelle job
  8)  Recupero dei file dat dai nodi
  9)  Eliminazioni dai nodi di lastdump e cartelle dx
 ----------
  d)  Gestione Database dei log
  r)  Gestione dei run in esecuzione
  x)  Gestione file DX e animazioni
```

base, è presente anche una vecchia versione, chiamata `manage_dump_old`. La versione più recente, comunque, è in grado di funzionare anche senza accesso al database, qualora fosse necessario.

```
 ----------
  i)  Info
 ----------
  q)  Uscita

Selezione:
```

Alcune delle opzioni rimandano ad altri menu, ed è anche possibile ottenere informazioni sulle opzioni da linea di comando (`Info`):

```
manage_dump ver.  1.29
Uso: manage_dump [opzioni]
Opzioni conosciute:
    --verbose      Scrive a schermo informazioni di debug
-q, --quite        Non scrive a schermo informazioni di debug
-r, --rename       Usa rename anziché mv per lo spostamento.
                   Non Ã¨ richiesto l'ordine esatto dei nodi,
                   ma non si ha output a schermo del comando.
-m, --mv           Usa mv anziché rename per lo spostamento.
                   Opzione di default. Come per la copia, l'elenco
                   dei nodi deve essere uguale a quello con cui
                   sono stati creati i dump.
-n, --nothing      Non esegue le operazioni, ma stampa solo a
                   schermo le linee di comando
    --noitape5     Non aggiorna il file itape5 col nuovo idrun
-v, --version      Stampa la versione dell'eseguibile.
-z, --zip          Comprime i file prima di trasferirli
--ssh              Usa ssh per eseguire comandi remoti (default)
--rsh              Usa rsh per eseguire comandi remoti
-user utente       Specifica il nome utente da utilizzare per le
                   operazioni, anziche' l'utente corrente.
-path percorso     Specifica un percorso alternativo dove si trova
                   il database. Path di default = ~/logs/
-administrator     Opzioni avanzate per gestione database e logs.
-cl                Abilita la modalità  'command line'
-h -cl             Richiama una pagina di aiuto sulle
                   opzioni di command line
-h, --help         Richiama questa pagina di aiuto
```

I sotto–menu per la gestione del database, dei run in esecuzione e dei file per DX sono riportati di seguito:

Gestione del database

```
********************** Manage  Dump **********************
************************ ver.  1.29 ************************
************************ 31.08.2006 ************************
-----------------
 Utente : francesco
-----------------

Gestione database dei log:
--------------------
  1)  Informazioni complete su un intervallo di idrun
  2)  Informazioni ristrette su un intervallo di idrun
  3)  Informazioni generali sul database
  4)  Cambio del database
  5)  Correzione di stato di uno o piu' idrun
  6)  Ricerca di job in esecuzione
  7)  Informazioni sulla "storia" di un run
 ----------
  o)  Cambio dell'unita' di output
 ----------
  q)  Menu principale

Selezione:
```

Gestione dei run in esecuzione

```
********************** Manage  Dump **********************
************************ ver.  1.29 ************************
************************ 31.08.2006 ************************
-----------------
 Utente : francesco
-----------------

Gestione dei run in esecuzione:
--------------------
  1)  Ricerca di job in esecuzione
  2)  Sospensione di un run in esecuzione
  3)  "Wake up" di un run in sospensione
  4)  Kill di un run in esecuzione o sospensione
```

```
 ----------
  q)  Menu principale

Selezione:
```

Gestione dei file DX e animazioni

```
********************** Manage  Dump **********************
************************ ver.  1.29 ************************
************************ 31.08.2006 ************************
-----------------
 Utente : francesco
-----------------

Gestione DX e animazioni:
---------------------
  1)  Informazioni sui file dx presenti in uno o piu' job
  2)  Copia su uno o più nodi una o più cartelle dx (animazioni)
  3)  Generazione ppm o filmati a partire da dati già sui nodi
  4)  Copia in locale di dati dx già  ripartiti sui nodi
  5)  Ripartizione sui nodi di dati dx locali
  6)  Merge dei file dx copiati in locale
 ----------
  q)  Menu principale

Selezione:
```

Non vengono qua descritte in dettaglio tutte le funzionalità del programma, ma solo le caratteristiche più significative. È opportuno notare, in ogni caso, che l'intero *manage_dump* è stato scritto in Fortran95 ed utilizza le stesse routine per l'accesso al database utilizzate da Kiva.

Scelta del dump da cui ripartire

In un run parallelo, ogni processo scrive la sua parte di dump. Per ripartire da un dump, è necessario un run con lo stesso numero di processi di quello che

lo ha generato, senza però che le macchine debbano essere necessariamente le stesse. Accedendo alle informazione memorizzate nel database, infatti, Kiva è in grado di rintracciare anche su altri nodi la parte di dump necessaria per ogni processo.

Un run di Kiva può avere scritto più dump, ovvero l'ultimo (`otape8`), il penultimo (`last_dump`), ed altri scritti a tempi o crank stabiliti. Prima di ripartire da un certo run, identificato da un numero di idrun, è necessario scegliere da quali di questi ripartire.

In *manage_dump* è presente l'opzione `Copia da otape8 a itape7`, che permette, una volta indicato l'idrun, di scegliere da quale dump ripartire. In caso si scelga un dump creato a tempi o crank stabiliti, viene chiesto anche il valore corrispondente. Non è possibile però sapere quali valori di tempo o crank erano state specificate in fase di esecuzione ed è dunque necessario esaminare l'itape5, reperibile nella directory di lavoro corrispondente (vedi paragrafo 3.7.1).

manage_dump non fa una copia del file richiesto, ma genera un hard link al file, per non occupare ulteriore spazio su disco. In caso di errore, viene riportato l'output del comando.

Ripartenza da dump con numero di processi diverso

Per cambiare il numero di processi di un dump parallelo sono stati sviluppati due programmi, basati su Kiva, chiamati `collectdump` e `dividedump`, che convertono rispettivamente un dump parallelo in seriale e viceversa. È possibile eseguirli da linea di comando o direttamente all'interno di *manage_dump*. Il primo di questi programmi è in grado di ricostruire un dump seriale a partire da un gruppo di file generati in parallelo, mentre il secondo si occupa del compito inverso, cioè suddividere un file per la ripartenza seriale in un insieme di dump paralleli. Sono in grado di gestire tutte le variabili di campo e tutte le informazioni salvate, così come le informazioni relative allo spray.

Recupero dei file DX

È possibile copiare, dalle directory locali delle macchine su cui un run è stato eseguito, una sola cartella DX o tutto il contenuto, compresi file dat, dump etc.... La destinazione può essere anche una macchina remota, purché si abbiano i permessi di scrittura nel percorso indicato.

Alla fine della copia, viene chiesto se eseguire `dxmerge`[11] (e `dropmerge`)

[11] `dxmerge` e `dropmerge` si occupano di ricostruire i file per DX, sia per il dominio di calcolo che per lo spray, rispettivamente, a partire dai file che ogni processo di un run parallelo ha generato.

sulle cartelle DX copiate. Questa operazione può anche essere eseguita in un secondo momento, anche non su tutte le cartelle, ma solo su una parte.

Eliminazione delle cartelle contenenti dump

L'opzione 9 di *manage_dump* permette di eliminare le directory locali di lavoro (vedi 3.7.1) relative ad un idrun o ad un intervallo, potendo specificare in quest'ultimo case se eliminare solo run temporanei (scratch) o tutti. L'intervallo viene specificato separando primo ed ultimo idrun con uno spazio.

Questa operazione non elimina la directory di lavoro, contenente l'eseguibile, i file di ingresso e quelli di uscita Per cancellare anche questa, anche in un secondo momento, è disponibile un'ulteriore opzione, accessibile solo in modalità *administrator*[12] per ragioni di sicurezza.

Gestione del database

Tramite questa opzione si accede ad un altro menu, in cui è possibile effettuare alcune operazioni di ricerca e visualizzazione di elementi del database, ottenere informazioni generali o eseguire operazioni di mantenimento, come sbloccare un database rimasto locked, cambiare il database da usare, scegliere un file come output delle query, etc.... Alcune di queste opzioni sono accessibili solo in modalità *administrator*.

[12]Questa modalità è attivabile eseguendo `manage_dump -admin`.

3.7 Esecuzione di Kiva

3.7.1 Modalità standard

Struttura delle directory

L'esecuzione di Kiva richiede un'appropriata struttura di directory ed adeguati permessi di lettura e scrittura per ogni utente che debba eseguirlo. In particolare si richiede:

- **Computer locale**

  Logs Directory `~/logs` in cui sono contenuti il database dei run, i valori di `md5sum` eseguito sui dump, per eseguire un controllo di integrità alla ripartenza, ed i file contenenti, per ogni compilazione eseguita, tutte le informazioni necessarie per ricostruire, tramite Subversion, i sorgenti da cui è stato ricavato un determinato eseguibile.

  File di input standard Directory `~/itapes` in cui sono contenuti i file di ingresso standard a cui faranno riferimento i link simbolici creati in fase di esecuzione da Kiva nella directory di lavoro. Questo percorso deve essere raggiungibile dai nodi su cui girano i vari processi del run parallelo.

  Directory di lavoro Directory che viene creata automaticamente dal-

lo script con cui viene eseguito Kiva. In particolare è del tipo `~/run/job.#idrun`, dove `#idrun` è un numero progressivo, attualmente a 5 cifre, che identifica in maniera univoca ogni esecuzione, indipendentemente dall'utente che la esegue. Il valore di idrun che verrà utilizzato alla prossima esecuzione è contenuto in uno specifico file che deve essere accessibile da tutte i computer da cui si lanciano gli script di esecuzione, ma non dai nodi.

- **Nodi di calcolo**

  Directory locale di lavoro Directory dove vengono scritti i dump, i file DX, il log (copia di quanto riportato nel database dei run) e i file dati. È composta come `$HOMELOCKIVA/`*`nomeutente`*`/job.#idrun`. Qualora la variabile d'ambiente `$HOMELOCKIVA` non sia definita, viene preso come default il valore `/home/`.

Script di esecuzione

Kiva deve essere lanciato tramite appositi script, in funzione del gestore delle risorse presente sul cluster in uso[13]. La sintassi per l'esecuzione può ovviamente variare a seconda delle specifiche esigenze. Un esempio è

```
startjob_pbs --np|-n #proc --exe|-e nomeexe [--input|-i input/dir]\
  [--options|-O opzioni] [--zip|-z compressore] [--add|-a file|dir]\
```

[13] Ad esempio, è stato scritto uno script che interagisce con PBS, chiamato `startjob_pbs`

```
[-I|--itape5 nomeitape5] [--help|-h] [--tipo|-t tiponodi]\
[--out|-o fileout]
```

Lo script, dopo aver controllato che sia stato specificato il nome dell'eseguibile ed il numero dei processi, ricava il valore di idrun da utilizzare, aggiornandolo per la successiva utilizzazione, crea la directory di lavoro `~/run/job.#idrun`, copia al suo interno l'eseguibile, il file `itape5` insieme tutti i file del tipo `itape??` presenti nella directory di input, eventualmente copia ulteriori file e directory specificati da linea di comando, si sposta nella directory di lavoro ed esegue Kiva, passando per linea di comando il valore di idrun corrente[14].

I file di ingresso, presenti nella cartella di input, che sono stati copiati nella directory di lavoro ma che non sono stati utilizzati, perché non richiesti[15] o perché letti da quelli standard, qualora specificato in itape5[16], vengono eliminati al fine di ridurre lo spazio di disco occupato.

Alla fine del run, l'eseguibile, i file di ingresso, escluso l'itape5, i file di uscita e l'eventuale reindirizzamento dell'output vengono compressi. Viene inoltre chiamato *manage_dump* per controllare che il run sia terminato in

[14]Questo permette di utilizzare lo stesso progressivo senza che i nodi di calcolo abbiano accesso diretto al file contenente il valore disponibile di idrun.

[15]Ad esempio, un itape17 in caso di ripartenza da dump

[16]Bloccheto #0047.

maniera controllata da Kiva, altrimenti lo stato del run, nel database, viene corretto da *Running* ad *Aborted*[17].

3.7.2 Modalità locale

Kiva può essere eseguito anche in modalità locale, in cui cioè l'unica directory di lavoro, sia per la creazione di dump e `DX`, sia per i file di dati, è la directory dove si trova l'eseguibile. È opportuno assicurarsi, quindi, che tutti i processi del run parallelo abbiano accesso alla directory di lavoro. La linea di comando per eseguire Kiva in questa modalità varia a seconda del gestore di risorse in uso. Un esempio, basato sull'impiego di PBS, è il seguente

```
mpiexec kivadyn_par -local [-idrun #idrun]
```

L'opzione `-local` impone a Kiva di utilizzare `./` come directory in cui cercare tutti i file di ingresso, compresi eventuali Dump da cui ripartire, e in cui scrivere qualsiasi file di output. L'opzione `[-idrun #idrun]` è invece facoltativa e serve per impostare il numero di idrun da utilizzare. Il valore di default è pari ad 1, ma è utile cambiarlo nel caso in cui si voglia ripartire da un dump generato con un precedente run in modalità locale. L'opzione `-local` inibisce in ogni caso l'accesso al database dei run effettuati.

[17] Vedere sezione 3.6 per maggiori dettagli.

Capitolo 4

Accoppiamento 3D – 1D e 0D

4.1 Accoppiamento 3D–0D

La condizione al contorno tra un condotto 3D ed un volume 0D, ovvero la sezione di efflusso della valvola, è stata implementata come una variante di una condizione al contorno di velocità, in cui la distribuzione di velocità ad ogni time step viene dedotta a partire da una simulazione interamente 3D di riferimento, mentre la componente media normale alla superficie di efflusso viene calcolata in maniera tale da rispettare il flusso di massa imposto dal volume 0D.

Più in dettaglio, viene inizialmente eseguita una simulazione con condotto e cilindro tridimensionali, così da memorizzare, per ogni valvola, i vettori velocità dei nodi di griglia che individuano la sezione di efflusso. Il profilo

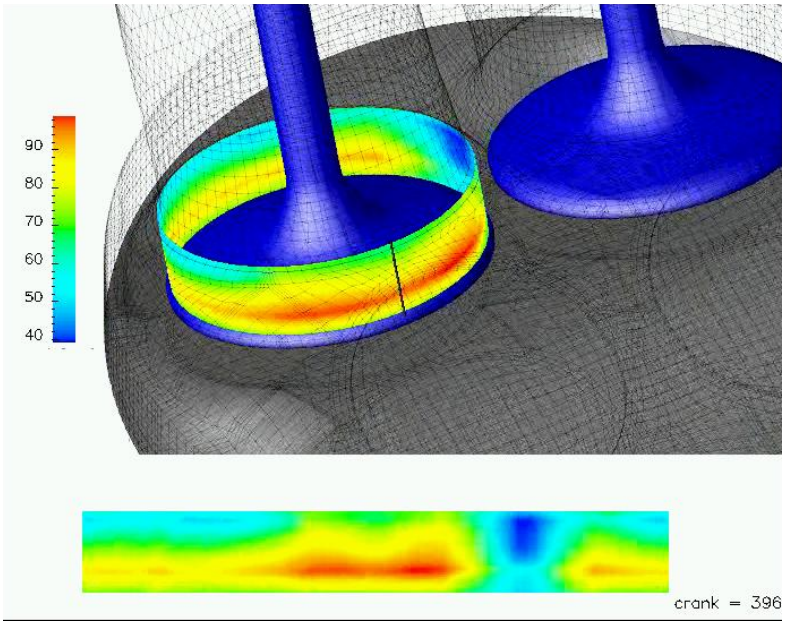


Figura 4.1: Esempio di profilo di velocità nella sezione di efflusso

di velocità nella sezione di efflusso può infatti assumere forme molto complesse, dipendenti sia dall geometria della valvola stessa, sia dai condotti e dalla camera di combustione, nonché dalla vicinanza con altre valvole, come mostrato in figura 4.1. Questi vettori vengono rappresentati, in coordinate cilindriche, in un sistema di riferimento avente l'asse verticale coincidente con la direzione di spostamento della valvola, mentre l'origine degli angoli è scelto come lo spigolo topologico in basso a sinistra, come mostrato in figura 4.2.

Le informazioni da memorizzare, per ogni valvola e per ogni scelto istante di output, hanno lunghezza variabile, principalmente perché il numero di

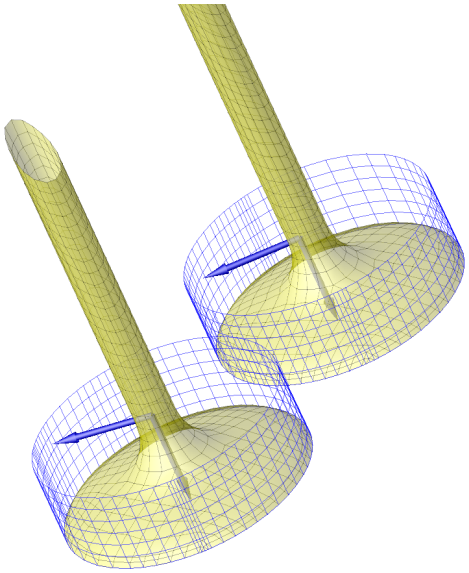


Figura 4.2: Sistema di coordinate locale per ciascuna valvola

piani di celle tra la valvola e la sua sede sulla testa del cilindro varia durante la simulazione. Quindi, al fine di minimizzare la quantità di memoria utilizzata per memorizzare questi campi di velocità di riferimento, è stato utilizzato una struttura dati definita opportunamente. Viene dichiarato, infatti, un vettore, con dimensione pari al numero di valvole presenti, composto da elementi di un tipo di dato derivato, composto a sua volta da delle liste record – puntatore di un secondo tipo derivato, avente al suo interno dei campi di dimensione variabile, sfruttando per questo alcune nuove caratteristiche degli standard Fortran95 e Fortran 2003. Una schematizzazione del tipo di dato utilizzato viene riportata in figura 4.3.

Durante l'esecuzione di un caso con condotto e valvola 3D accoppiati ad

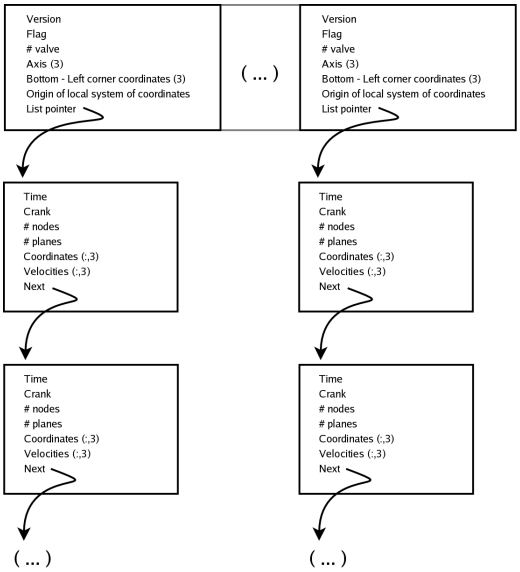


Figura 4.3: Struttura di dati per accoppiamento 3D–0D

un cilindro zero – dimensionale, per ogni boundary 3D – 0D usato, Kiva richiede un file generato da un simulazione tridimensionale completa, come sopra descritto, contenente le informazioni sul campo di moto, uno sfasamento tra le alzate valvole omologhe nelle simulazioni 3D e 0D, nonché un modello 0D per la descrizione del cilindro. I campi di moto di riferimento vengono interamente caricati in memoria all'inizio della simulazione, trasformando le velocità, se necessario, nel sistema di riferimento locale della valvola corrispondente e normalizzando i vettori rispetto alla componente radiale media.

Ad ogni time step, poi, utilizzando una routine parallela ricorsiva per tener conto del più generico partizionamento dei domini, vengono calcolati i

valori medi delle proprietà termodinamiche di cui il modello 0D ha bisogno, passandoli poi al modello 0D vero e proprio, che a sua volta calcola il flusso di massa evolvente in quel time step richiesto. Le velocità dei nodi al contorno, quindi, vengono imposte in maniera tale che la velocità normale media su tutta la sezione di efflusso della valvola soddisfi la portata richiesta, mentre le componenti di ogni vettore velocità vengono dedotte interpolando, nello spazio e nel tempo, i valori ricavati dal campo di riferimento.

La forma del campo di velocità risulterà dunque analoga a quanto era stato ottenuto durante la simulazione 3D, mantenendo le caratteristiche del flusso, particolarmente importante in alcuni tipi di sezioni di ingresso che generano un moto di *swirl*. Questo permette di simulare, in minor tempo ma con buona confidenza dei risultati, una sequenza di cicli di aspirazione e scarico su un solo cilindro, o perfino interi sistemi di aspirazione, quando una rappresentazione monodimensionale del moto nella zona intorno alle valvole comporterebbe, per le geometrie coinvolte, errori troppo significativi. Si rimanda al paragrafo 5.1 per una descrizione dei risultati ottenuti con l'applicazione di questa strategia.

4.2 Accoppiamento 3D–1D

Nello sviluppo della metodologia di accoppiamento tra il codice 3D ed il 1D, è stata posta molta attenzione per ottenere uno strumento flessibile e comodo per modellare problemi di grandi dimensioni, aumentando il livello di dettaglio là dove le geometrie presenti lo richiedano.

Si è scelto di seguire una strategia di accoppiamento esplicito alle condizioni al contorno. Sia nel codice 3D che in quello 1D sono state introdotte delle nuove *boundary conditions* in grado di gestire la comunicazione delle informazioni necessarie tra le diverse zone del dominio. In particolare, come mostrato in figura 4.4, ad ogni time step la componente 3D del programma valuta i valori delle quantità termodinamiche che devono essere utilizzate come condizioni al contorno per la componente 1D. Il codice monodimensionale, quindi, avanza di un time step 3D, suddividendo se necessario l'intervallo in più parti[1], ed imposta le condizioni al contorno per la griglia 3D utilizzando i valori dei nodi più vicini alla giunzione. La sezione 3D riprende poi il controllo dell'esecuzione fino all'iterazione successiva.

[1] È possibile specificare in input il numero massimo di intervalli 1D per ogni intervallo 3D. Il valore dovrebbe essere grande abbastanza da ridurre il numero di calcoli 3D, solitamente molto dispendiosi, ma non troppo, in modo da non introdurre instabilità numeriche. La scelta ottimale varia da caso a caso, ma un valore attorno a 4 sembra essere valido nella maggior parte dei problemi affrontati.

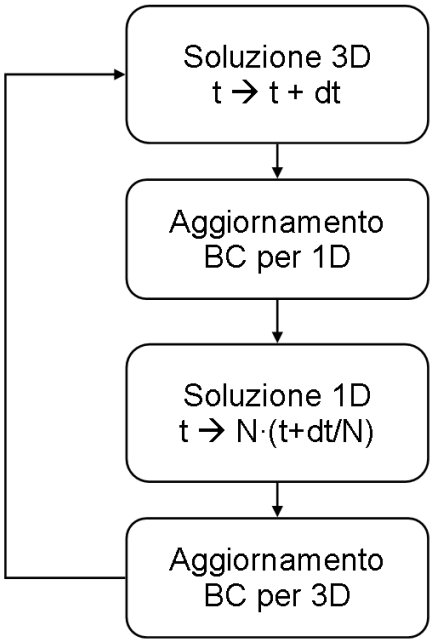


Figura 4.4: Sequenza delle operazioni per ogni iterazione

Il codice 1D utilizza una discretizzazione spaziale del secondo ordine ed ha quindi due nodi esterni al dominio di calcolo. Per catturare al meglio i fenomeni transitori nelle superfici di interfaccia, i valori di tali nodi vengono calcolati come media dei primi due *strati* di celle del dominio 3D[2], come schematizzato in figura 4.5. Tutti i valori necessari, come pressione, temperatura, velocità, composizione chimica, variabili per il modello di turbolenza, etc. . . , vengono quindi mediati negli ultimi due strati di celle 3D, separatamente, ed applicati ai nodi esterni al dominio durante l'applicazione delle condizioni al

[2]Si ricorda che la mesh di calcolo è strutturata.

contorno 1D per il time step in corso. Viceversa, alla fine del calcolo 1D, i valori delle variabili nell'ultimo nodo interno al dominio monodimensionale vengono applicati uniformemente come condizione al contorno per il codice tridimensionale.

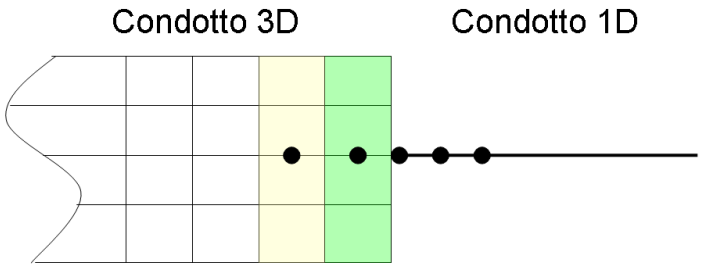


Figura 4.5: Schematizzazione delle nuove condizioni al contorno

Questa strategia, resa possibile dalle molte novità introdotte in Kiva, in particolare dalla completa riscrittura e generalizzazione della gestione delle condizioni al contorno, ha fatto sì che sia ora possibile collegare un numero arbitrario di elementi monodimensionali ad un ugual numero di *boundary* di un arbitrario numero di griglie tridimensionali, senza preoccuparsi della direzione del flusso evolvente. È naturalmente opportuno collegare elementi 3D e 1D là dove si suppone che il flusso sia sufficientemente uniforme sulla sezione, altrimenti l'approssimazione monodimensionale non è più rispettata.

È inoltre estremamente semplice sostituire un elemento mono o zero dimensionale con la sua rappresentazione 3D, una volta generata la griglia, e

viceversa. Questa caratteristica permette da un lato di valutare gli effetti del flusso tridimensionale in una parte del modello, confrontando due simulazioni uguali, dove la zona in esame sia una volta 3D e l'altra no, dall'altro di raffinare la soluzione a passi successivi aggiungendo via via parti tridimensionali nei punti in cui le approssimazioni di un codice monodimensionale possono ridurre la qualità della soluzione. Nel caso, ad esempio, di un motore automobilistico con sistema di ricircolo dei gas di scarico, usato come caso test e i cui risultati sono riportati nel paragrafo 5.3.1, è possibile iniziare l'analisi dal solo modello monodimensionale, veloce da simulare, per poi sostituire l'*airbox* con la sua rappresentazione 3D, così da studiare il differente riempimento dei cilindri, per poi sostituire la valvola EGR zerodimensionale con uno o più modelli 3D, così da misurare l'effettivo grado di ricircolo per diverse condizioni di funzionamento, e così via.

L'integrazione fra i due modelli, mono e tridimensionali, è dunque completa e robusta, sfruttando le stesse metodologie di input dei programmi originali, avendo solamente aggiunto la possibilità di gestire questi nuovi tipi di condizioni al contorno e di giunzioni.

L'ultimo importante aspetto da gestire riguarda il calcolo in parallelo. Il codice 1D utilizzato è strettamente sequenziale e molto esteso, mentre molti sforzi sono stati spesi sulla versione in uso di Kiva per renderla funzionante ed efficiente in calcolo distribuito, ed era questa una caratteristica irrinun-

ciabile per una integrazione che fosse effettivamente utilizzabile su problemi ad elevato numero di celle, come le attuali applicazioni richiedono.

La prima valutazione ha dunque riguardato il tempo necessario ad effettuare la parte di calcolo monodimensionale in confronto alla tridimensionale per un problema di interesse industriale. Si è scelto dunque un motore per uso sportivo[3] in cui l'*airbox* viene modellato tramite una griglia 3D costituita da circa 300.000 celle, mentre il resto del motore, compresi i dieci cilindri ed i relativi processi di combustione, vengono simulati dal codice monodimensionale. Si è potuto dunque osservare che, in una esecuzione seriale, la parte 3D del calcolo occupa più del 98% del tempo totale, mentre il restante 2% è impegnato dal codice 1D.

Si è deciso dunque di non parallelizzare il codice monodimensionale, poiché, ai fini di una sua applicazione in accoppiamento con Kiva, il guadagno di prestazione possibile era ridotto e non avrebbe giustificato lo sforzo necessario per una sua pressoché intera riscrittura.

Una possibile strategia per il funzionamento in parallelo era la possibilità di far eseguire la parte di calcolo 1D ad uno solo dei processi, cui assegnare una minore porzione 3D, per evitare *overloading* dannosi ai fini dell'efficienza. I valori per le condizioni al contorno, subito prima e subito dopo la parte 1D

[3]Si veda il paragrafo 5.3.2.

del calcolo, dovevano poi essere scambiate tra i nodi di calcolo per poter procedere correttamente per l'iterazione successiva.

Visti però i tempi di calcolo in gioco, si è deciso di far eseguire un identico calcolo 1D a tutti i processi di una esecuzione parallela, risparmiando quindi la necessità di sincronizzazione dei dati ottenuti da questa porzione del processo di simulazione. È facile capire come il peso del calcolo 1D aumenti all'aumentare del numero di processi, ma, come si può vedere in figura 4.6, questo rimane comunque limitato.

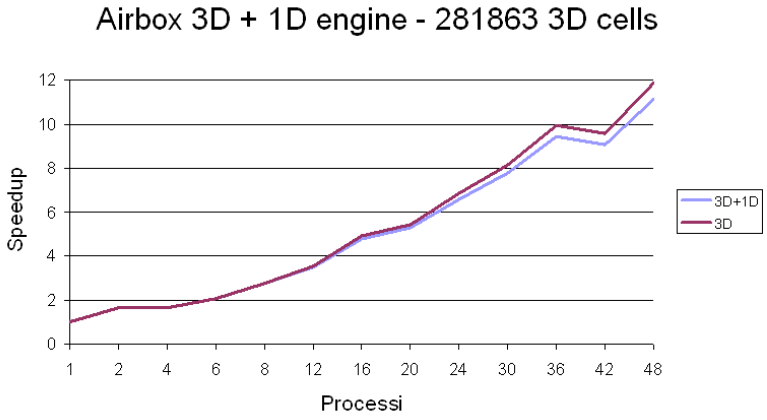


Figura 4.6: Speedup 3D–1D

L'ultima caratteristica implementata, ai fini di una completa integrazione tra i due codici, è stata una efficiente integrazione del sistema di ripartenza da una soluzione intermedia. Le informazioni necessarie a riprendere la simulazione 1D sono state aggiunte a quelle già esistenti per la parte 3D e

vengono scritte all'interno dello stesso file nel corso della simulazione. Nessun ulteriore accortezza è quindi necessaria quando si voglia continuare una simulazione da una condizione raggiunta in precedenza.

Il risultato è una singola applicazione che, insieme all'accoppiamento 3D–0D descritto al paragrafo 4.1, permette di gestire problemi completi e complessi, consentendo un differente livello di dettaglio per parti differenti, e sfruttando i vantaggi del calcolo parallelo, per quanto riguarda la dispendiosa simulazione 3D, al fine di ridurre significativamente il tempo di calcolo o di aumentare le dimensioni delle geometrie tridimensionali presenti nel problema. Si rimanda ai paragrafo 5.2 e 5.3 per una descrizione dettagliata dei risultati ottenuti.

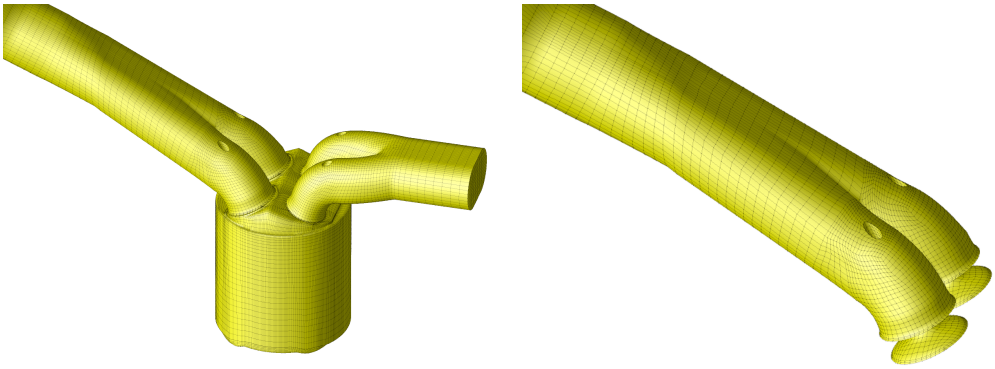
Capitolo 5

Risultati

5.1 Accoppiamento 3D–0D

Al fine di verificare la nuova condizione al contorno introdotta per collegare Kiva ad un modello zerodimensionale, è stato esaminato il flusso attraverso un condotto di aspirazione, sia con la presenza del cilindro che senza. Il confronto è stato effettuato facendo attenzione a non avere reflusso dal cilindro alla valvola durante la simulazione 3D, per escludere eventuali disturbi legati al modello 0D introdotto.

Come descritto nel paragrafo 4.1, è necessaria una simulazione completamente tridimensionale al fine di ottenere i campi di velocità attraverso le sezioni di efflusso delle valvole. L'esecuzione di riferimento è stata condotta durante la fase di aspirazione del cilindro con quattro valvole rappresentato



(a) Modello interamente 3D (b) Condotto di aspirazione 3D

Figura 5.1: Griglie di calcolo impiegate per la validazione del modello di accoppiamento 3D–0D

in figura 5.1a, a partire da poco dopo il punto morto superiore fino a circa 50° dopo la chiusura della valvola di aspirazione.

Lo stesso condotto di aspirazione, senza cilindro né sezione di scarico, è stato quindi scelto per verificare la strategie di accoppiamento, come mostrato in figura 5.1b. La condizione al contorno all'ingresso del condotto di aspirazione è la stessa del caso completo, ma viene imposta la nuova condizione al contorno di velocità alla sezione di efflusso della valvola, in cui il campo di moto è ricavato dalla prima simulazione.

Un confronto fra le due simulazioni, in termini di pressione e temperature medie nel condotto durante la simulazione, viene riportato nelle figure 5.2 e 5.3. È significativa, inoltre, la differenza in termini di tempo di calcolo, di

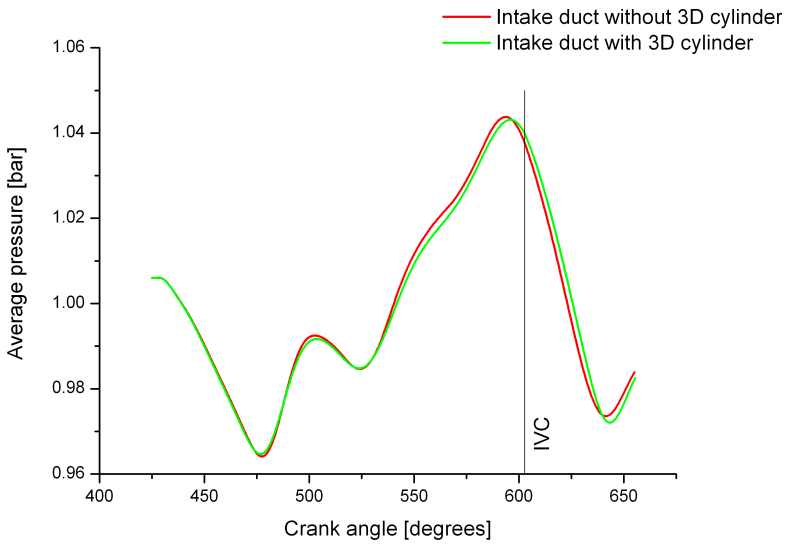


Figura 5.2: Confronto tra i valori medi di pressione nel condotto di aspirazione per le simulazioni con e senza la presenza del cilindro

seguito riportata.

	Tempo trascorso (min)
Simulazione 3D completa	188.81
Condotto 3D senza cilindro	23.84

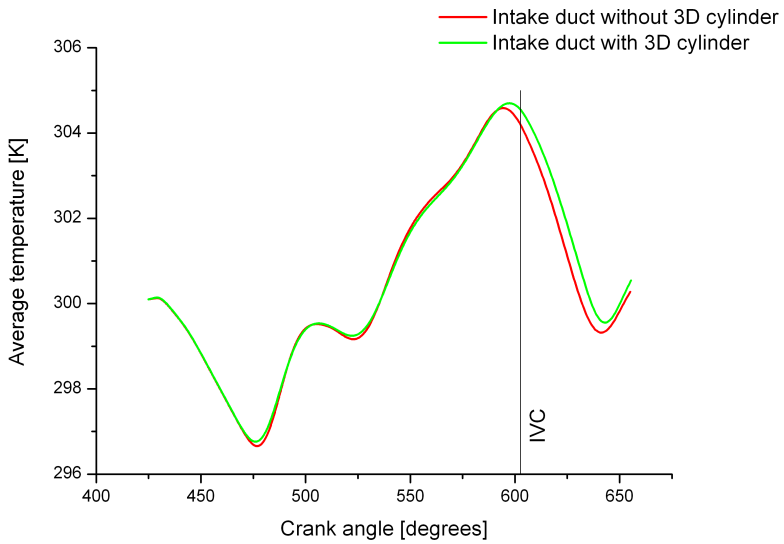


Figura 5.3: Confronto tra i valori medi di temperatura nel condotto di aspirazione per le simulazioni con e senza la presenza del cilindro

5.2 Accoppiamento 3D–1D

5.2.1 Condotto 1D–3D con condizioni al contorno costanti

Il primo test per la strategia di accoppiamento 3D–1D descritta al nel paragrafo 4.2 è il flusso attraverso un condotto, composto da tre settori, come mostrato in figura 5.4. Il diametro non varia lungo l'intero condotto e le connessioni con i volumi alle estremità sono supposte essere senza perdite. Si suppone, inoltre, che il condotto sia adiabatico e senza attrito.

Il tubo viene rappresentato sia con tutti gli elementi 1D, sia con uno 3D,

Figura 5.4: Test case 1D–3D–1D

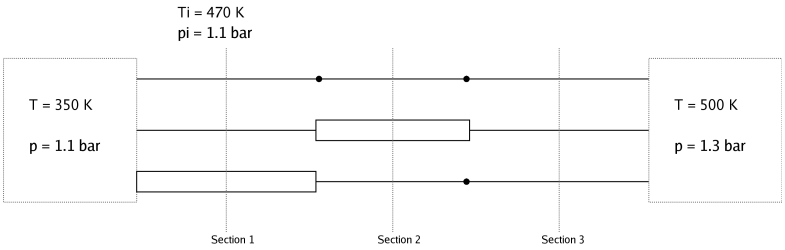
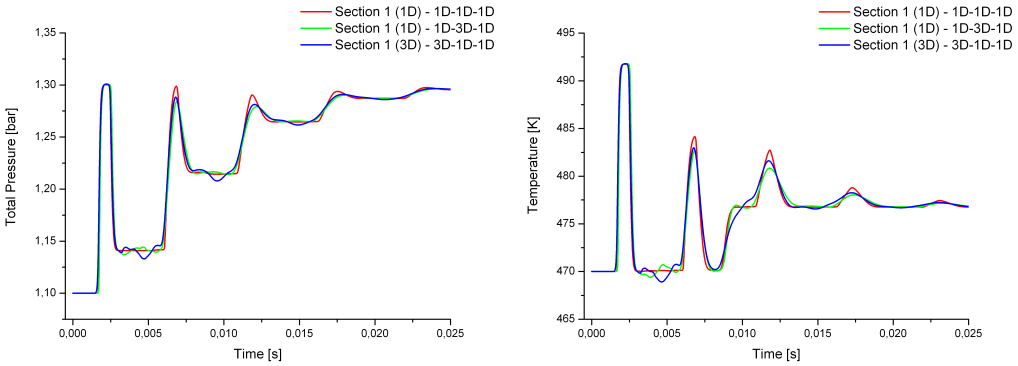


Figura 5.5: Differenti configurazioni

al centro o ad una estremità, come mostrato in figura 5.5. Il confronto tra le soluzioni è stato effettuato tenendo conto della sezione mediana di ogni elemento.

Si consideri, inizialmente, una condizione al contorno di pressione totale e temperatura costanti nel tempo ad entrambe le estremità del condotto. Come mostrato in figura, i valori imposti sono:



(a) Pressione totale

(b) Temperatura

Figura 5.6: Risultati delle simulazioni solo 1D e con tratti 3D, con condizioni al contorno costanti.

$$\begin{aligned} T_1 &= 350\,\mathrm{K} \\ p_1 &= 1.1\,\mathrm{bar} \\ T_2 &= 500\,\mathrm{K} \\ p_2 &= 1.3\,\mathrm{bar} \end{aligned} \tag{5.1}$$

mentre le condizioni iniziali all'interno sono:

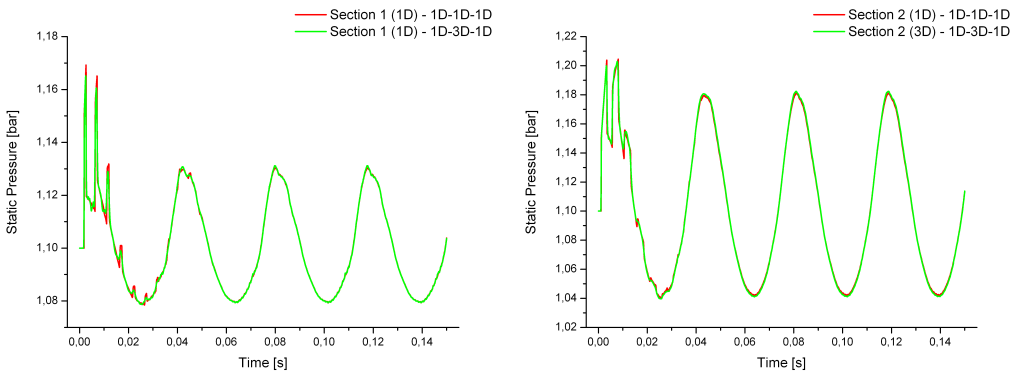
$$\begin{aligned} T_i &= 470\,\mathrm{K} \\ p_i &= 1.1\,\mathrm{bar} \end{aligned} \tag{5.2}$$

I valori di pressione totale e temperatura ottenuti durante il transitorio sono rappresentati nelle figure 5.6a e 5.6b, rispettivamente.

5.2.2 Condotto 1D–3D con condizioni al contorno variabili

Si modifichi ora la condizione al contorno all'estremità destra del condotto in figura 5.5, imponendo una pressione totale variabile data dalla relazione:

$$p_0 = 1.15 + 0.3 \cdot sin\left(\frac{1}{T} \cdot t\right) \tag{5.3}$$

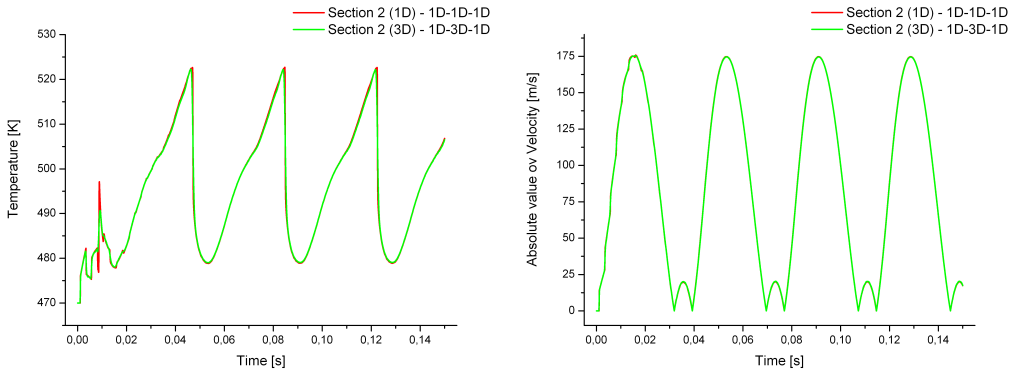


(a) Sezione 1

(b) Sezione 2

Figura 5.7: Risultati, in termini di pressione statica, delle simulazioni solo 1D e con tratti 3D, con condizioni al contorno variabili sinusoidalmente.

I risultati ottenuti sono riportati nelle figure 5.7a – 5.8b, in termini di pressione, temperatura e velocità nelle prime due sezioni del condotto. È importante notare come, durante la simulazione, il vettore velocità cambi direzione, quindi si inverte il flusso di massa attraverso i boundary sia con i volumi esterni, sia tra le sezioni 1D e 3D.



(a) Temperatura

(b) Modulo velocità

Figura 5.8: Risultati, in termini di temperatura e velocità nella sezione 2, delle simulazioni 1D e 1D–3D con condizioni al contorno variabili. Si vede, dal grafico delle velocità, come il flusso inverta il suo moto nel corso della simulazione.

5.2.3 Analisi di riduzione del rumore

Un'interessante applicazione di analisi accoppiata 3D – 1D riguarda la predizione del rumore aerodinamico delle sue componenti. La metodologia può essere applicata sia al sistema di aspirazione (come in [29]), sia al silenziatore posto allo scarico, essendo queste le due principali sorgenti di pressione sonora verso l'esterno del veicolo. L'utilizzo del solo modello 1D potrebbe infatti non essere in grado di cogliere gli effetti d'onda presenti nelle componenti in esame, che influenzano il livello di pressione sonora in uscita e le frequenza per le quali si misurano i valori massimi e minimi,

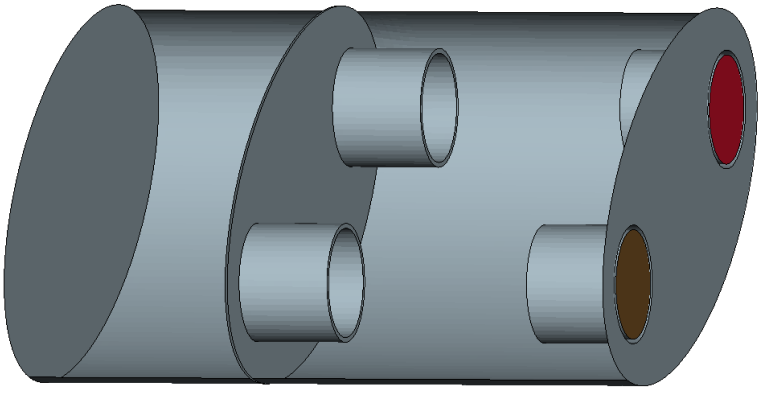


Figura 5.9: Silenziatore

Si è proceduto quindi all'analisi del rumore trasmesso attraverso un silenziatore, la cui geometria è rappresentata in figura 5.9, mentre la rappresentazione della mesh di calcolo utilizzata, la cui dimensione complessiva è pari

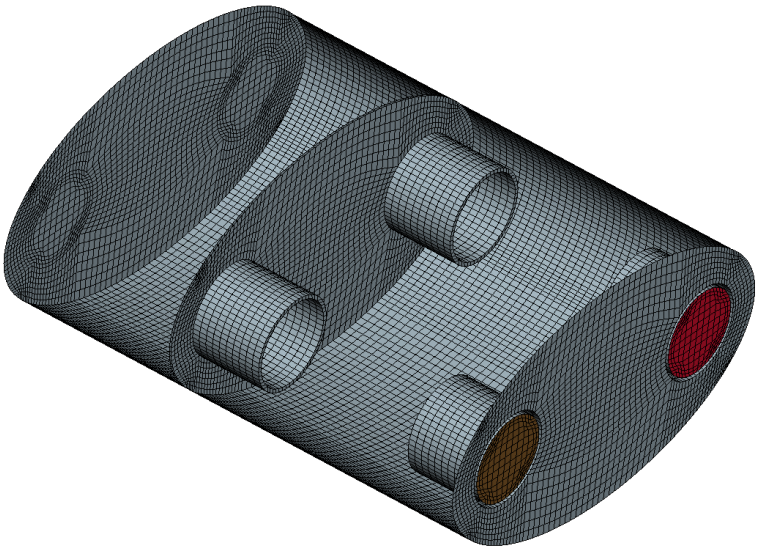


Figura 5.10: Silenziatore – Mesh di calcolo superficiale

a 138168 celle, è riportata in figura 5.10[1].

Le due parti di ingresso ed uscita, di colore diverso nelle figure, sono state collegate a due condotti 1D di uguale sezione e lunghezza arbitraria. Come condizioni al contorno è stata assegnata in ingresso un'onda di pressione rappresentante un *rumore bianco*, riportata in figura 5.11. L'uscita è stata invece considerata connessa ad un ambiente a pressione costante pari al valor medio del segnale di entrata.

Il segnale di pressione allo scarico, misurato in un punto interno al corrispondente condotto 1D, rappresenta la risposta del sistema. Il valore di *noise reduction* è definito come il rapporto tra le pressioni sonore a monte e quella

[1]La mesh di volume, essendo strutturata, dipende esclusivamente da quella superficiale.

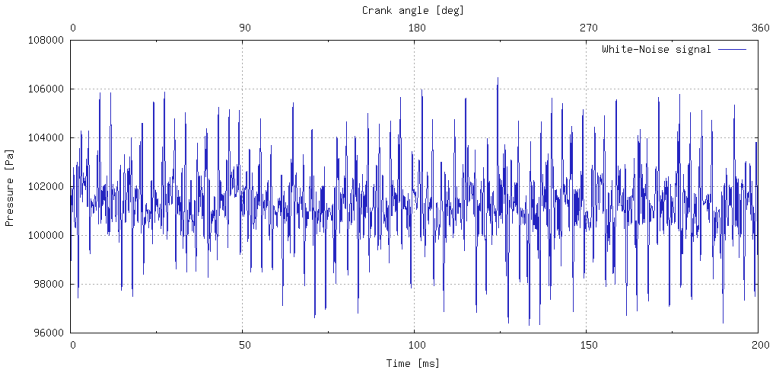


Figura 5.11: Segnale di pressione di rumore bianco

a valle del silenziatore, in dB.

$$NR = 10 \log_{10} \left(\frac{P_u}{P_d} \right) \tag{5.4}$$

dove

$$P_u = \frac{\bar{p}_u^2}{\rho \, c} \, \Omega_u \tag{5.5}$$

$$P_d = \frac{\bar{p}_d^2}{\rho \, c} \, \Omega_d \tag{5.6}$$

Nelle precedenti relazioni, i pedici u e d indicano, rispettivamente, i valori a monte (*upstream*) ed a valle (*downstream*) del silenziatore, $\bar{p}$ indica la media quadratica del segnale di pressione, ρ la densità dell'aria, c la velocità del suono e Ω l'area di efflusso.

Sostituendo le relazione 5.6 e 5.5 nella 5.4, si ha:

$$NR = 20 \log_{10} \left(\frac{\bar{p}_u}{\bar{p}_d} \right) + 10 \log_{10} \left(\frac{\Omega_u}{\Omega_d} \right) \tag{5.7}$$

Descrivendo il segnale di pressione nel dominio delle frequenze per mezzo di una trasformazione di Fourier, l'ampiezza Δp di ogni armonica definisce il livello di pressione sonora del segnale alla data frequenza come:

$$SPL = 20 \log_{10} \left(\frac{\Delta p}{p_{ref}} \right) \tag{5.8}$$

Combinando la 5.7 con la 5.8 e trascurando il contributo molto piccolo proveniente dalla variazione di sezione Ω_u/Ω_d si ottiene:

$$NR \cong SPL_u - SPL_d \tag{5.9}$$

I livelli di pressione sonora a monte e a valle del silenziatore per le frequenze fino a 2000 Hz sono riportati in figura 5.12 mentre il valore della riduzione del rumore è rappresentato in figura 5.13. È interessante osservare come per alcune frequenze il valore della pressione sonora in uscita sia maggiore di quello in ingresso.

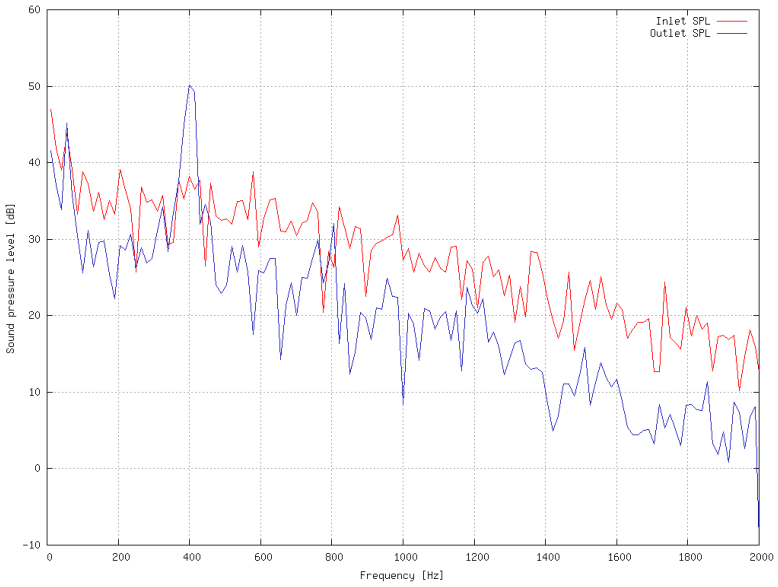


Figura 5.12: Livello di pressione sonora a monte ed a valle del silenziatore

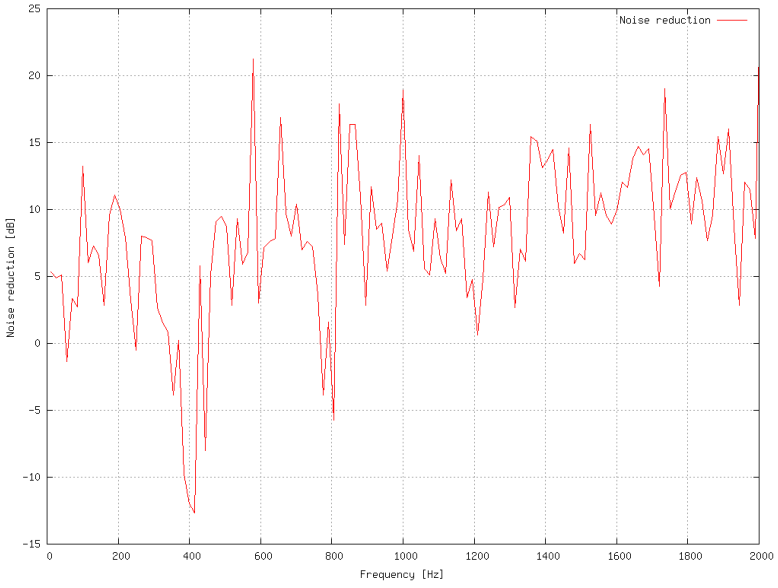


Figura 5.13: Riduzione del rumore

5.3 Simulazione 3D-1D di interi sistemi MCI

5.3.1 MCI automobilistico con EGR

Al fine di validare il codice con un caso più esaustivo, si è optato per la simulazione di un intero motore automobilistico ad accensione comandata, il cui schema è riportato in figura 5.14.

Si tratta di un motore 4 cilindri, 16 valvole, con EGR nella linea di scarico. Sono stati confrontati i risultati ottenuti sia con una simulazione interamente 1D–0D, sia sostituendo il plenum d'aspirazione zerodimensionale (*P4* nello schema) con un modello 3D, riportato in figura 5.15. Il plenum ha sei porte, ognuna connessa ad un condotto monodimensionale. Più in particolare, la prima porta connette il plenum con la linea di aspirazione, il secondo con il sistema EGR, mentre gli ultimi quattro lo collegano con i cilindri. Vale la pena evidenziare come non ci siano limiti al numero di giunzioni che colleghino la griglia 3D con elementi 1D, sia di boundary 3D comunque caratterizzati, e che si possono collegare più elementi 3D a una o più linee 1D.

Mentre nella simulazione interamente 1D – 0D i cilindri hanno tutti esattamente lo stesso comportamento, ovvero lo steso valore di riempimento, frazione di EGR, pressione media e così via, introducendo il plenum 3D ci si aspetta di ottenere differenti valori delle variabili nei diversi cilindri, a seconda della geometria.

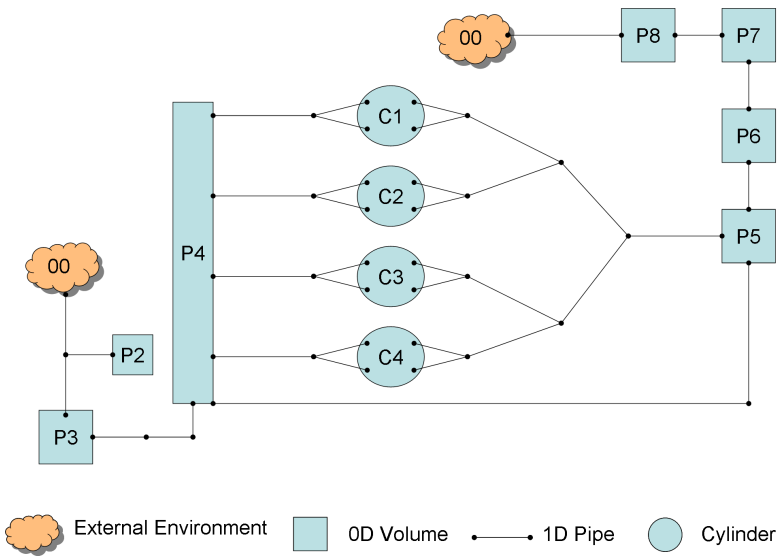


Figura 5.14: Schema 1D del motore completo

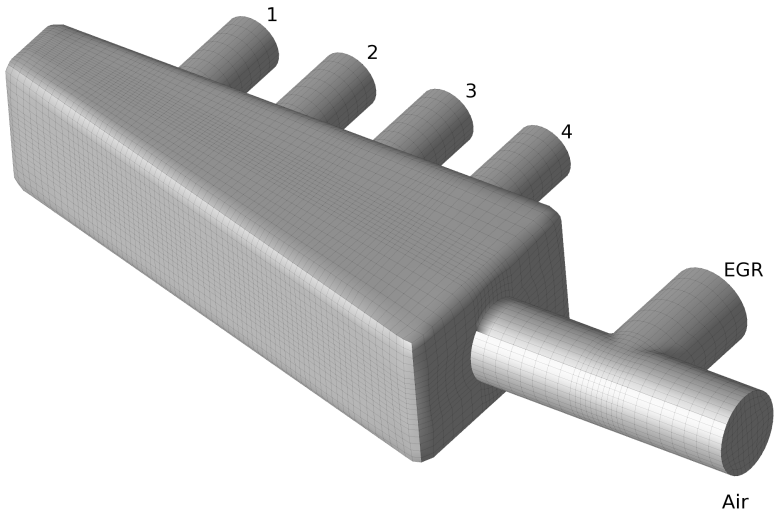
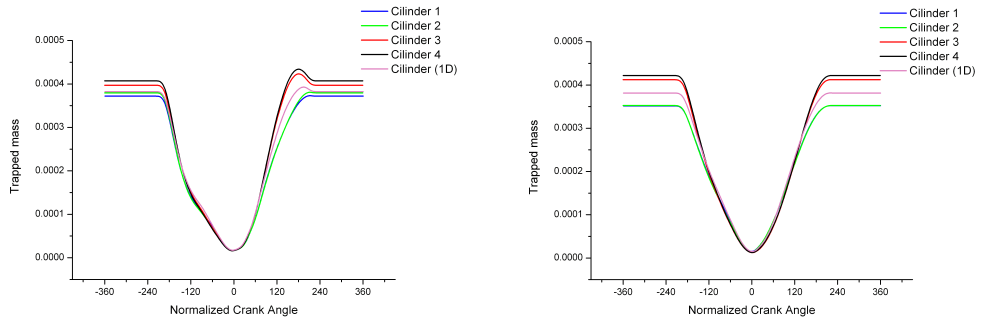


Figura 5.15: Plenum di aspirazione 3D



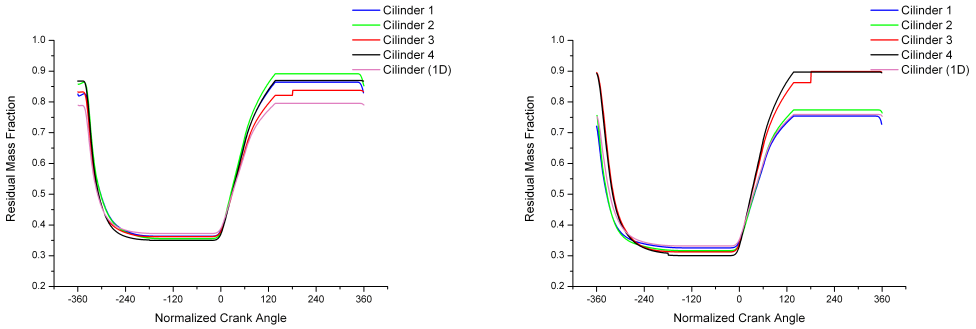
(a) 3000 rpm

(b) 5000 rpm

Figura 5.16: Massa intrappolata nei cilindri

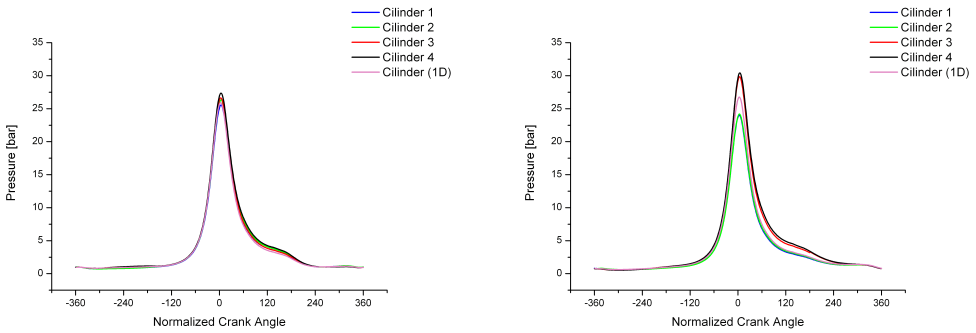
Come mostrato nelle figure 5.16 e 5.17, la quantità di massa intrappolata e la frazione di residui della combustione è differente nei quattro cilindri, scostandosi in maniera non trascurabile dai valori della simulazione 1D.

I valori di pressione e temperatura, per entrambe le velocità di rotazione in esame, confrontati con quanto ottenuti dal modello 1D, sono rappresentate nelle figure 5.18 – 5.19.



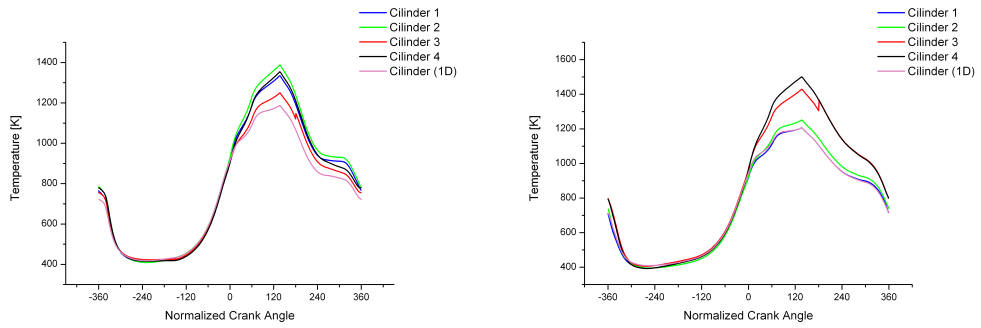
(a) 3000 rpm (b) 5000 rpm

Figura 5.17: Frazione di gas combusti



(a) 3000 rpm (b) 5000 rpm

Figura 5.18: Pressione nei cilindri



(a) 3000 rpm (b) 5000 rpm

Figura 5.19: Temperatura nei cilindri

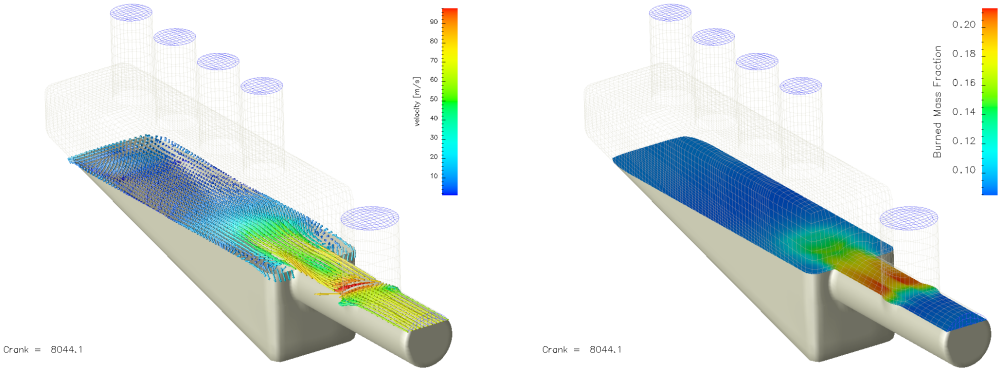


Figura 5.20: Velocità e frazione di EGR – Sezione 1

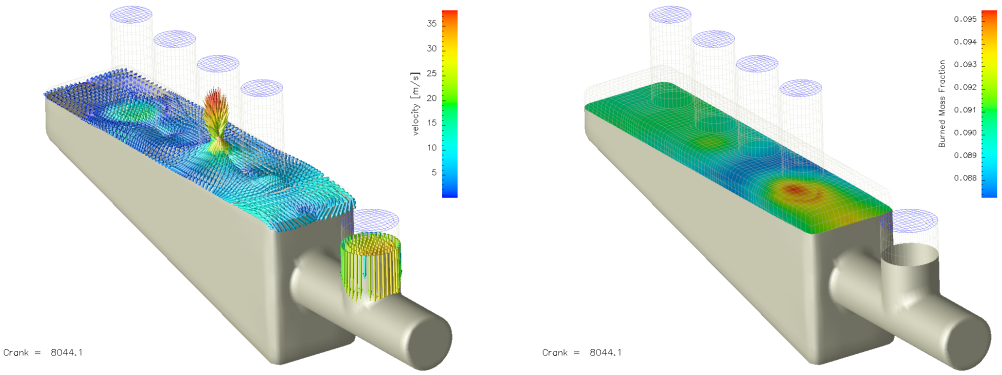


Figura 5.21: Velocità e frazione di EGR – Sezione 2

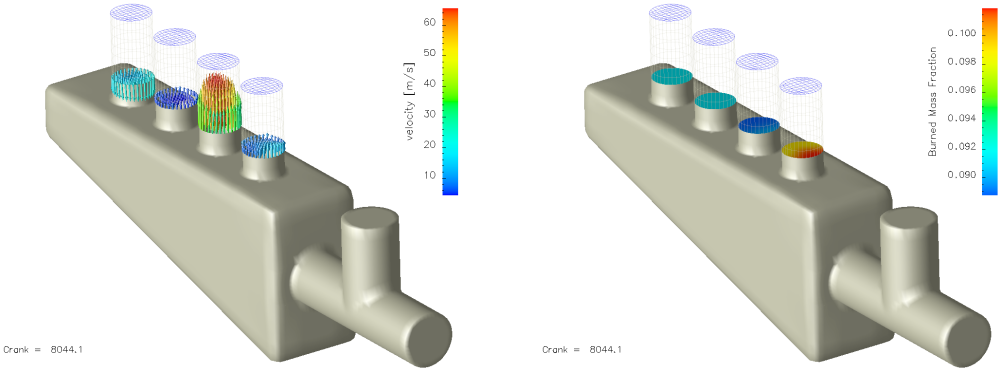


Figura 5.22: Velocità e frazione di EGR – Sezione 3

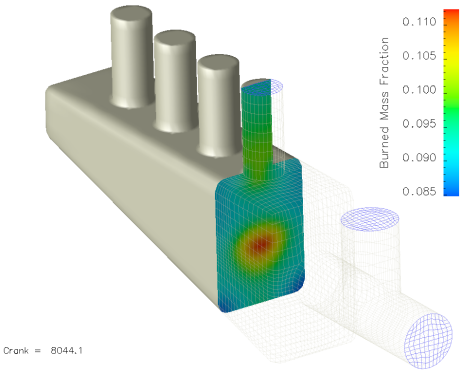


Figura 5.23: Frazione di gas combusti

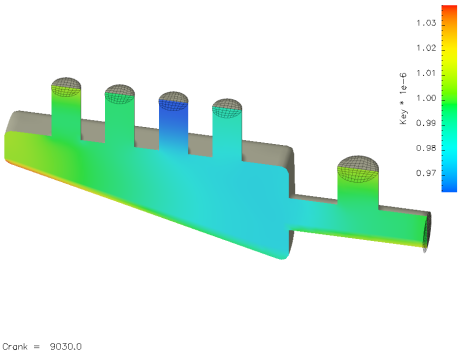


Figura 5.24: Pressione

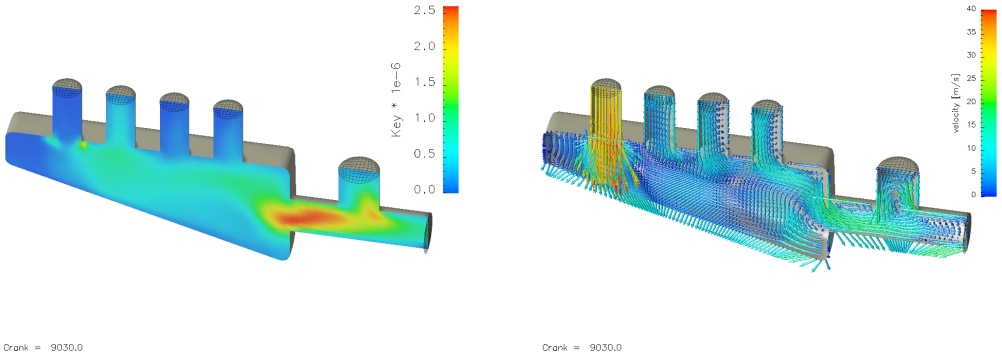


Figura 5.25: Energia cinetica turbolenta e campo di velocità

5.3.2 MCI automobilistico ad alte prestazioni

In figura 5.26 e 5.27, invece, viene mostrato un esempio di applicazione in un airbox di un motore sportivo. Si tratta in particolare di una unità 10 cilindri di tipo *Formula 1*, secondo il regolamento in vigore fino al campionato 2005. Il layout del motore è piuttosto semplice, non essendo presenti filtri, silenziatori o molti dei componenti presenti nelle normali unità di serie. Essendo però un motore di tipo aspirato, gli effetti d'onda presenti nell'airbox a diversi regimi di rotazione sono fondamentali per predire e controllare il riempimento di tutti i cilindri e di conseguenza la potenza erogata. La velocità di rotazione presa in esame è pari a 13000 rpm.

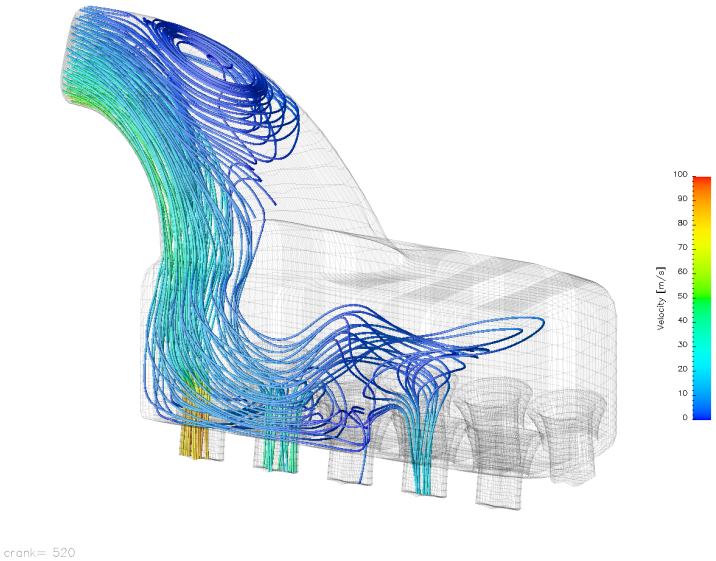


Figura 5.26: Linee di flusso all'interno dell'airbox – Motore sportivo

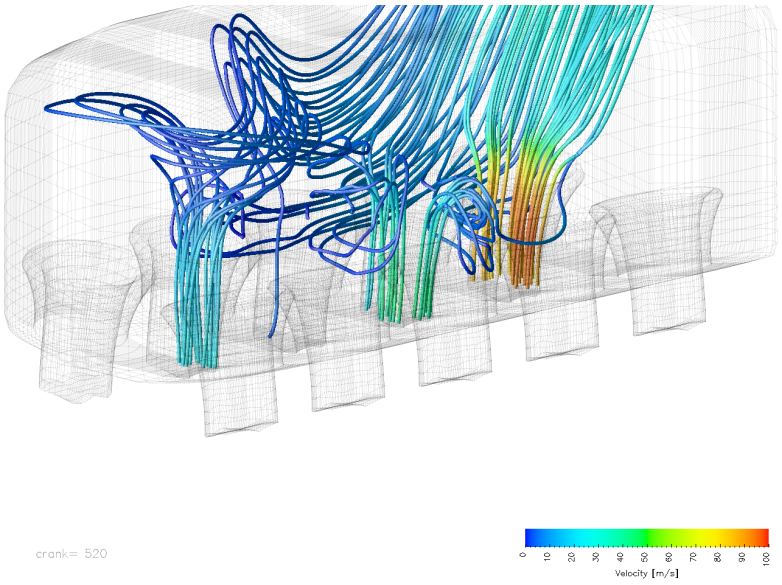


Figura 5.27: Linee di flusso all'interno dell'airbox – Motore sportivo

Conclusioni

Nel corso dei precedenti capitoli sono stati illustrati in dettaglio gli strumenti utilizzati, le modifiche apportate ad essi e le novità introdotte al fine di ottenere un codice di calcolo in grado di simulare in dettaglio ed accuratamente un intero motore a combustione interna.

Per quanto riguarda il codice di calcolo tridimensionale, si è visto come la sostituzione dei solutori per le equazioni differenziali che descrivono il problema fisico abbia da sola portato un sensibile incremento della velocità di esecuzione, senza intaccare la qualità della soluzione. La successiva parallelizzazione di ogni suo componente ha poi permesso di ridurre ulteriormente il tempo di attesa per il calcolo di una soluzione per mezzo della suddivisione del dominio tra i vari processi.

Allo stesso tempo, molte nuove caratteristiche sono state implementate al fine di estendere i campi di utilizzo del programma, di renderlo più flessibile e più facilmente modificabile in futuro. Inoltre la robustezza ed il traccia-

mento delle cause di una esecuzione fallita sono stati migliorati. Molti sforzi sono stati fatti per inserire il software all'interno di un ambiente di lavoro flessibile e allo stesso tempo automatizzato, facendo sì che la gran parte delle operazioni per l'esecuzione e per il trattamento dei dati sia ora accessibile per mezzo di un'interfaccia, in maniera da ridurre il tempo impiegato in compiti come il post–processing di una soluzione, la produzione di filmati, il mantenimento dello spazio disco, la gestione delle esecuzioni in corso e passate, il cambiamento del numero di processi su cui eseguire un run parallelo, e così via.

Una particolare attenzione è stata prestata ad uno degli aspetti più dispendiosi in termini di tempo durante la preparazione di un nuovo motore, ovvero la configurazione dell'algoritmo di riposizionamento dei nodi della griglia per adattarsi allo spostamento di tutte le superfici mobili. Questo è stato quindi esteso a tutte le zone interessati da spostamenti di componenti, quindi cilindro e condotti di aspirazione e scarico, ed è stato generalizzato in maniera da permettere una completa configurazione attraverso il file di ingresso, non più per mezzo di una parte di codice appositamente scritta. Questo permette di ridurre le modifiche al codice sorgente solo per introdurre nuove caratteristiche e consente inoltre un notevole risparmio di tempo nel corso della configurazione dell'algoritmo per un nuovo motore. Queste informazioni, inoltre, essendo scritte nel file di ingresso standard, sono sempre

disponibili, automaticamente selezionate e facilmente modificabili, qualora sia necessario.

Sono state poi descritte le strategie di accoppiamento del codice di calcolo tridimensionale con un generico modello zero–dimensionale da un lato, e con un completo e validato codice mono–dimensionale dall'altro.

In particolare, l'integrazione completa tra modelli 3D e 1D ha permesso di ottenere uno strumento completo per la simulazione di sistemi complessi in grado di descrivere con diversi gradi di dettaglio differenti zone di interesse in modo da sfruttare al meglio le caratteristiche di entrambi.

I risultati presentati mostrano l'efficacia del metodo utilizzato ed esplorano alcuni dei possibili campi di applicazione, come la progettazione di un sistema di aspirazione al fine di massimizzare il riempimento in tutti i cilindri o l'analisi del rumore acustico prodotto da un sistema di scarico.

Appendice A

Struttura dei buffer di dati

Vengono qui descritte le strutture dei buffer di spedizione per i dati in virgola mobile ed interi nelle nuove routine che gestiscono la deformazione dinamica della griglia di calcolo.

Questi buffer vengono utilizzati per trasmettere le informazioni sui pesi da utilizzare per l'algoritmo di *rezoning*. I valori scelti dall'utente vengono specificati nella sezione corrispondente del file di ingresso, come dettagliatamente descritto nel paragrafo 3.5, vengono letti dal processo *root* ed utilizzati per costruire la struttura dati rappresentata in figura 3.6 a pagina 81. Tali informazioni devono però essere comunicate agli altri processi, in una esecuzione parallela. Data la complessità della struttura di dati utilizzata, necessaria per garantire una sufficiente flessibilità di utilizzo, i dati devono essere *impacchettati* in due buffer, uno per i valori in virgola mobile ed un'altro per gli

interi, prima di poter essere agevolmente spediti ed interpretati dai riceventi.

Tabella A.1: Buffer dei dati reali

Zsquish				
1	2	...	$2 \cdot nzsq - 1$	$2 \cdot nzsq = i_{2,0}$
$crank_1\,(1)$	$crank_1\,(2)$	...	$crank_{nzsq}\,(1)$	$crank_{nzsq}\,(2)$
Rezpent				
$i_{2,0}+1$	$i_{2,0}+2$	$i_{2,0}+3$	$i_{2,0}+4$	$i_{2,0}+5$
$crank_0\,(1)$	$crank_0\,(2)$	$kloc_{0,1}$	$val_{0,1,1}$	$val_{0,1,2}$
...	$i_{2,0}+3+n_{0,1}$	$i_{2,0}+4+n_{0,1}$	...	$i_{2,0}+2+\sum_{i=1}^{18}(n_{0,i}+1) = i_{2,1}$
...	$val_{0,1,n_{0,1}}$	$kloc_{0,2}$	...	$val_{0,18,n_{0,18}}$
$i_{2,1}+1$	$i_{2,1}+2$	$i_{2,1}+3$	$i_{2,1}+4$	$i_{2,1}+5$
$crank_1\,(1)$	$crank_1\,(2)$	$kloc_{1,1}$	$val_{1,1,1}$	$val_{1,1,2}$
...	$i_{2,1}+3+n_{1,1}$	$i_{2,1}+4+n_{1,1}$	...	$i_{2,1}+2+\sum_{i=1}^{18}(n_{1,i}+1) = i_{2,2}$
...	$val_{1,1,n_{1,1}}$	$kloc_{1,2}$	...	$val_{1,18,n_{1,18}}$
$\vdots$				
$i_{2,nrez}+1$	$i_{2,nrez}+2$	$i_{2,nrez}+3$	$i_{2,nrez}+4$	$i_{2,nrez}+5$
$crank_{nrez}\,(1)$	$crank_{nrez}\,(2)$	$kloc_{nrez,1}$	$val_{nrez,1,1}$	$val_{nrez,1,2}$
...	$i_{2,nrez}+3+n_{nrez,1}$	$i_{2,nrez}+4+n_{nrez,1}$	...	$i_{2,nrez}+2+\sum_{i=1}^{18}(n_{nrez,i}+1)$
...	$val_{nrez,1,n_{nrez,1}}$	$kloc_{nrez,2}$	...	$val_{nrez,18,n_{nrez,18}}$

Tabella A.2: Buffer dei dati interi

Intestazione				
1	2	3	...	20
$nzsq$	$nrez$	$n_{0,1}$	...	$n_{0,18}$
21	...	38	...	$2 + 18 \cdot nrez = i_1$
$n_{1,1}$	...	$n_{1,18}$	...	$n_{nrez,18}$
Zsquish				
$i_1 + 1$	$i_1 + 2$	$i_1 + 3$	...	$i_1 + 8$
zsq_1	reg_1	$estremi_1\,(1)$	...	$estremi_1\,(6)$
$i_1 + 9$	...	$i_1 + 16$	...	$i_1 + 16 \cdot nzsq = i_{2,0}$
$flag_1\,(1)$	...	$flag_1\,(6)$	...	$flag_{nzsq}\,(6)$
Rezpent				
$i_{2,0} + 1$	$i_{2,0} + 2$	...	$i_{2,0} + 8$	$i_{2,0} + 9$
zsq_0	$str_{0,1,1}\,(1)$	...	$str_{0,1,1}\,(7)$	$pos_{0,1,1}\,(1)$
$i_{2,0} + 10$	$i_{2,0} + 11$	...	$i_{2,0} + 1 + \sum_{i=1}^{18} (9 \cdot n_{0,i}) = i_{2,2}$	
$pos_{0,1,1}\,(2)$	$str_{0,1,2}\,(1)$	...	$pos_{0,18,n_{0,18}}\,(2)$	
$\vdots$				
$i_{2,nrez} + 1$	$i_{2,nrez} + 2$	...	$i_{2,nrez} + 8$	$i_{2,nrez} + 9$
zsq_{nrez}	$str_{nrez,1,1}\,(1)$	...	$str_{nrez,1,1}\,(7)$	$pos_{nrez,1,1}\,(1)$
$i_{2,nrez} + 10$	$i_{2,nrez} + 11$	...	$i_{2,nrez} + 1 + \sum_{i=1}^{18} (9 \cdot n_{nrez,i}) = i_{2,2}$	
$pos_{nrez,1,1}\,(2)$	$str_{nrez,1,2}\,(1)$	...	$pos_{nrez,18,n_{0,18}}\,(2)$	

Bibliografia

[1] Amsden, A.A., *KIVA 3: A KIVA with Block Structured Mesh for Complex Geometry*, 1992, Los Alamos National Laboratory report LA - 12503-Ms.

[2] Amsden, A.A., *KIVA 3-V: A Block Structured KIVA Program for Engine with Vertical or Canted Valves*, 1997, Los Alamos National Laboratory report LA - 13313-Ms.

[3] Courant R., Friederichs K.O., Lewy H., *Uber die partiellendifferenz–leichungen der mathematischen phisik*, Mathematische Annalen 1928, vol. 100, pagg. 32–74. English Translation in *IBM journal* 1967, pagg. 215–234.

[4] Bozza F., Cameretti M.C., Tuccillo R., *Numerical Simulation of In–Cylinder Processes and Duct Flow in a Light–Duty Diesel Engine*, Fourth Int. Symp. on Small Diesel Engines, Varsavia, 1996, in "Journal of POLISH CIMAC", Vol. 2, n. 1, pp. 51–66, 1996.

[5] Bozza F., Tuccillo R., de Falco D., *A Two–Stroke Engine Model Based on Advanced Simulation of Fundamental Processes*, SAE paper 952139, also in "Design and Emissions of Small Two– and Four–Stroke Engines", SAE SP-1112, pp. 87–98, 1995.

[6] Bozza F., Gimelli A., Senatore A., Caraceni A., *A Theoretical Comparison of Various VVA Systems for Performance and Emission Improvements of SI Engines*, SAE Paper 2001-01-0671, 2001, also in "Variable Valve Actuation 2001", SAE SP-1599, ISBN 0-7680-0746-1.

[7] Bozza F., Cardone M., Gimelli A., Senatore A., Tuccillo R., *A Methodology for the Definition of Optimal Control Strategies of a VVT–Equipped SI Engine* proc. of. 5th Int. Conf. ICE2001 Internal Combustion Engines: Experiments and Modeling, September 2001.

[8] Bozza F., Gimelli A., Tuccillo R., *The Control of a VVA Equipped SI–Engine Operation by Means of 1D Simulation and Mathematical Optimization*, SAE Paper 2002-01-1107, 2002 SAE World Congress, Detroit, March 2002, also in "Variable Valve Actuation 2002", SAE SP-1692, ISBN 0-7680-0960-X. Published also on SAE 2002 Transactions - Journal of Engines.

[9] Bozza F., Torella E., *The Employment of a 1D Simulation Model for A/F*

Ratio Control in a VVT Engine, SAE Paper 2003-01-0027, 2003 SAE World Congress, Detroit, March 2003, also in "Variable Valve Actuation 2003".

[10] S. Filippone, P. D'Ambra, M. Colajanni: *Using a Parallel Library of Sparse Linear Algebra in a Fluid Dynamics Applications Code on Linux Clusters.* Parallel Computing - Advances & Current Issues, G. Joubert, A. Murli, F. Peters, M. Vanneschi eds., Imperial College Press Pub. (2002), pp. 441–448.

[11] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh *Basic Linear Algebra Subprograms for FORTRAN usage.* ACM Trans. Math. Soft., 5:308–323, 1979

[12] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson *An extended set of FORTRAN Basic Linear Algebra Subprograms* ACM Trans. Math. Soft., 14:1–17, 1988

[13] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson *Algorithm 656: An extended set of FORTRAN Basic Linear Algebra Subprograms* ACM Trans. Math. Soft., 14:18–32, 1988

[14] JJ. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling *A set of Level 3 Basic Linear Algebra Subprograms* ACM Trans. Math. Soft., 16:1–17,

1990

[15] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson *Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms* ACM Trans. Math. Soft., 16:18–28, 1990

[16] S. Filippone and M. Colajanni. *PSBLAS: A library for parallel linear algebra computation on sparse matrices.* ACM Trans. Math. Soft., 26(4):527–550, December 2000.

[17] G. Bella, A. Buttari, A. De Maio, F. Del Citto, S. Filippone, F. Gasperini, *FAST-EVP: an Engine Simulation Tool*, Proceedings of HPCC 2005.

[18] G. Bella, F. Bozza, A. De Maio, F. Del Citto, S. Filippone, *An Enhanced Parallel Version of Kiva–3V, Coupled with a 1D CFD Code, and Its Use in General Purpose Engine Applications*, in "High Performance Computing and Communications", Proceedings of HPCC 2006, Springer Berlin / Heidelberg, pag. 11–20, 2006.

[19] A. Buttari, P. D'Ambra, D. di Serafino and S. Filippone: *Extending PSBLAS to Build Parallel Schwarz Preconditioners*, in Proceedings of PARA'04, to appear.

[20] P. D'Ambra, D. Di Serafino, and S. Filippone: *On the Development of PSBLAS-based Parallel Two-level Schwarz Preconditioners*, Preprint n. 1/2005, Dipartimento di Matematica, Seconda Università di Napoli, 2005.

[21] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Pub., Boston, 1996.

[22] N. M. Nachtigal, S. C. Reddy, L. N. Threfethen: *How fast are non-symmetric matrix iterations?*, SIAM J. Matrix Anal. Applic., 13(1992), 778-795.

[23] L. N. Threfethen: *Pseudospectra of linear operators*, SIAM Review, 39(1997), 383–406.

[24] Harten A., *On a Class of High Resolution Total Variation Stable Finite Difference Schemes*, Jrl of Computational Physics, Vol. 21, 21–23, 1984.

[25] Manna M., *A Three Dimensional High Resolution Compressible Flow Solver*, PhD Thesis, Catholic Univ. of Louvain - Von Karman Institute for Fluid Dynamics, 1992, also in TN 180, VKI, 1992.

[26] Miller D.S., *Internal Flow Systems*, Second Edition, BHR Group Limited, 1990.

[27] Matthews R.D., Chin Y.W., *A Fractal–Based SI Engine Model: Comparisons of Predictions with Experimental Data*, SAE Paper 910075, 1991.

[28] Bozza F., Gimelli A., Siano D., Torella E. Mastrangelo G., *A Quasi–Dimensional Three–Zone Model for Performance and Combustion Noise Evaluation of a Twin–Spark High–EGR Engine* SAE Paper 2004-01-0619, SAE World Congress, Detroit, March 2004, also in "Modeling of Spark–Ignition Engines", SAE SP-1830, ISBN 0-7680-1366-6, pp. 63–73.

[29] Bozza F., Gimelli A., Piazzesi R., Fortunato F., Pianese V., Siano D., *The prediction of the Performance and Gasdynamic Noise Emitted by a Medium–Size Spark–Ignition Engine by Means of 1D and 3D Analyses*, SAE Paper 2007-01-0380, 2007 SAE World Congress, Detroit, April 2007.