DOTTORATO DI RICERCA

in

SCIENZE COMPUTAZIONALI E INFORMATICHE

Ciclo *XXIII*

Consorzio tra Università di Catania, Università di Napoli Federico II,

Seconda Università di Napoli, Università di Palermo,Università di Salerno

SEDE AMMINISTRATIVA: UNIVERSITÀ DI NAPOLI FEDERICO II

Raffaele Farina

Tecniche di Precondizionamento Inverso in un Modello Oceanico Globale

---

TESI DI DOTTORATO DI RICERCA

Alla Mia Famiglia e ad Angela.

I risultati più concreti della mia ricerca

sono stati il vostro infinito sostegno...

Sommario

Molte importanti applicazioni di interesse scientifico e industriale sono sempre riconducibili a sistemi dinamici basati sulle equazioni differenziali alle derivate parziali, tra queste ritroviamo anche i modelli oceanici globali.
I modelli oceanici sono una componente dei modelli climatici e sono in crescente sviluppo essendo utilizzati, non solo per lo studio delle dinamiche oceaniche, ma per l'intero sistema climatico globale.
I modelli in questione forniscono strumenti per l'interpretazione delle osservazioni oceaniche, la valutazione di scenari futuri come ad esempio quelli legati al riscaldamento climatico indotto dall'uomo e infine strumenti per le previsioni delle dinamiche degli oceani su differenti scale spaziali e temporali.
I modelli oceanici si basano sulle equazioni primitive della fluidodinamica e le variabili di interesse sono il campo delle velocità, la temperatura, la salinità, la pressione e la densità del fluido oceanico. Le principali differenze tra i vari modelli attualmente in uso sono nella parametrizzazione di fenomeni fisici di piccola scala, nel calcolo del gradiente di pressione superficiale, nell'adozione del sistema di coordinate verticali, nella scelta delle griglie orizzontali per la rappresentazione delle quantità fisiche del modello e infine nell'uso di diversi metodi numerici per la discretizzazione delle equazioni evolutive nelle equazioni fondamentali della fluidodinamica.
Nei modelli oceanici le scale temporali di interesse variano da mesi ad anni, ma nelle simulazioni alcuni dei fenomeni richiedono scale temporali che vanno da secondi a ore. Inoltre le simulazioni dei modelli vengono eseguite sull'intero pianeta e non si conosce se vi siano luoghi sulla Terra dove la

risoluzione delle griglie è meno importante. Conseguentemente i costi computazionali di questi modelli sono enormemente grandi tanto da limitare l'uso a volte di alcuni di questi.

La proposta di tesi si inserisce in questo contesto, con l'obiettivo di affrontare nello specifico le problematiche numeriche relative ad un solutore iterativo, il Gradiente Coniugato (CG), utilizzato per la risoluzione di un sistema lineare, derivante dal nucleo ellittico di un modello per la circolazione oceanica globale chiamato NEMO (Nucleus for European Modeling of the Ocean) introdotto per "chiudere" il modello alle equazioni primitive . Un simile sistema ovvero un sistema avente sempre la stessa matrice incompleta è risolto ad ogni passo temporale del modello oceanico.

Sperimentalmente si è osservato una lenta convergenza del solutore iterativo CG nel modello quando le risoluzioni delle griglie di discretizzazioni aumentano o quando il dominio di simulazione include luoghi in prossimità dei poli terrestri. Il sistema lineare, $Ax = b$, che si vuole esaminare è ottenuto dal nucleo ellittico del modello oceanico globale NEMO per mezzo delle differenze finite nello spazio. La sua matrice dei coefficienti A gode della proprietà di sparsità, di simmetria e definita positività. Per avere un' idea delle dimensioni di questo sistema si fa riferimento ad ORCA025, il quale è uno delle principali configurazioni del modello, in cui l'ordine di grandezza per la matrice incompleta A è di 10^6.

La presente tesi evidenzia in principio le cause principali della scarsa rapidità di convergenza del solutore dovute, in primo luogo, all'aumento della dimensione del sistema causato dall' incremento della risoluzione delle griglie

spaziali. Un'analisi più approfondita, invece, rivela che questa insufficienza della velocità di convergenza è provocata anche dalla variazione del rapporto delle funzioni α e β rappresentanti i coefficienti del problema ellittico in esame.

E' noto che, l'uso dei precondizionatori nei metodi iterativi si rivela essere fondamentale per aumentare la rapidità di convergenza di questi nella risoluzione di un sistema lineare $Ax = b$, con delle definite proprietà. L'obiettivo è ottenere un'approssimazione della soluzione, nei noti spazi di Krylov, nel minor numero di iterazioni possibile al fine di ridurre il tempo di risoluzione e gli effetti della aritmetica a precisione finita quando questi vengono eseguiti su calcolatore. Il precondizionatore P utilizzato nel modello oceanico NEMO è quello diagonale ovvero $P = diag(A)$.

All'interno della tesi è presentato un primo risultato che lega la distanza $\|P^{-1}A - I\|_F$ alla dimensione di A e al rapporto tra le funzioni α e β nella norma di Frobenius mediante una stima di un limite inferiore ed uno superiore per la grandezza in disamina . Questo risultato caratterizza appunto la lenta convergenza del CG quando la dimensione del sistema lineare aumenta e quando il modello è risolto in prossimità dei poli terrestri a causa di uno sbilanciamento tra le funzioni α e β.

Essendo A una matrice simmetrica e definita positiva, si introduce nella tesi il precondizionatore $\widehat{P} = U^\top U$ ottenuto dalla fattorizzazione incompleta di Cholesky con U fattore sparso di Cholesky. La decomposizione incompleta di Cholesky è un potente precondizionatore per problemi simmetrici, definiti positivi e con elevata sparsità. Si dimostra nel nostro caso, attraverso es-

perimenti numerici, come questo precondizionatore sia in grado di annullare gli effetti della aumentata dimensione del sistema lineare $Ax = b$, a seguito dell'intensificazione delle griglie di discretizzazione e dello sbilanciamento delle funzioni α e β nelle adiacenze dei poli. Questo precondizionatore ha la facoltà di incrementare notevolmente la rapidità di convergenza del solutore iterativo nel sistema lineare $Ax = b$ del modello oceanico NEMO. Purtroppo per grandi dimensioni di A, lo svantaggio principale dell'uso di $\widehat{P}$ sta nella risoluzione di un sistema lineare, di matrice incompleta $\widehat{P}$, eseguita ad ogni iterazione del solutore iterativo ed alla sua difficoltà di parallelizzazione.

Per superare questa problematica lo studio si è rivolto ad un' altra classe di precondizionatori, chiamati precondizionatori inversi sparsi, i quali cercano di approssimare in modo diretto l'inversa di A. La loro implementazione richiede all'interno del solutore l'impiego dell'algoritmo del matrice per vettore sparso per ogni iterazione del solutore.

Su questa nuova classe di precondizionatori, grazie a delle assunzioni sulla matrice A ed a delle stime pervenute sugli elementi dell'inversa U^{-1} del fattore di Cholesky U di A, si è costruito un nuovo precondizionatore $\widehat{P}'$ ad hoc *per il problema dato, capace di ridurre il tempo di risoluzione del PCG con precondizionatore diagonale nel caso dello sbilanciamento dei coefficienti α e β del* 25%.

Lo studio di opportuni precondizionatori che accelerano la rapidità di convergenza dei solutori iterativi si inserisce, in modo significativo, anche nel contesto futuro che vede le comunità climatiche utilizzare le nuove architetture di computing, formate da sistemi ibridi con GPU (Graphical Processing

Unit) a core paralleli e CPU (Control Processor Unit) multi-core. L' utilizzo delle GPU NVIDIA, consente di trarre, in maniera economica, un enorme vantaggio grazie alla loro elevata capacità di elaborazione parallela, mediante l'utilizzo di centinaia di core, completamente programmabili e in grado di eseguire migliaia di thread congiuntamente. Nella tesi, viene anche presentata un'applicazione per GPU NVIDIA, sviluppata in ambiente CUDA (Compute Unified Device Architecture) e CUBLAS (CUDA Basic Linear Algebra), che implementa il metodo del PCG (Gradiente Coniugato Precondizionato) in singola precisione con precondizionatore diagonale P, per la risoluzione del nucleo ellittico in disamina. In termini di efficienza, per gli ordini di grandezza della dimensione del problema sperimentati, fissato il valore della tolleranza sul residuo alla singola precisione, la versione parallela del PCG ottenuta mediante CUDA e CUBLAS si è dimostrata capace di ridurre i tempi di elaborazione dell' algoritmo sequenziale fino a 6 volte con l' hardware utilizzato.

Infine il motivo principale della proposta dell'utilizzo di precondizioatori di tipo inversi sparsi per il sistema lineare del modello è dovuto non solo al fatto che hanno una elevata capacità di ridurre il numero di iterazioni del solutore per raggiungere la convergeza, ma anche perchè la loro implementazione, richiede il calcolo del matrice per vettore ad ogni iterazione del solutore. Questa operazione è caratterizzata da un elevato grado di parallelismo sui dati che la rende particolarmente adatto all' elaborazione parallela mediante CUDA. L'uso dei precondizionatori inversi sparsi nel modello ha ridotto il tempo di risoluzione del PCG con precondizioantore diagonale su GPU fi-

no all' $87,5\%$ *nel caso dello sbilanciamento dei coefficienti* α *e* β *del nucleo ellittico.*

I risultati pervenuti in questa tesi vogliono essere una proposta all'uso di tecniche di precondizionamento più efficienti nei modelli oceanici, al fine di superare i problemi di scarsa convergenza dei metodi iterativi all'interno dei modelli oceanici globali. Inoltre, anche in vista del futuro uso, da parte della comunità scientifica oceanografica, delle rivoluzionarie architetture di calcolo parallele GPU NVIDIA, i risultati in questo lavoro anticipano l'uso di precondizionatori inversi sparsi al fine di sfruttare al meglio il tipo di parallelismo di tali architetture di calcolo.

Si termina ribadendo che accelerare i solutori iterativi nei modelli oceanici globali è un obiettivo fondamentale nel quadro futuro della modellazione oceanica per spingere oltre nel tempo le simulazioni per le previsioni Climatiche.

Acronimi

BLAS Basic Linear Algebra Subprograms

CBLAS C Basic Linear Algebra Subprograms

CG Conjugate Gradient

COO Coordinate List

CPU Control Processor Unit

CRS Compressed Row Storage

CSC Compress Storage Column

CUBLAS CUDA Basic Linear Algebra Subprograms

CUDA Computer Achitecture United Device

DRAM Dynamic Random Access Memory

GFDL Geophysical Fluid Dynamics Laboratory

GPU Graphic Processing Unit

HIM Hallberg isopicnica Model

HOPS Harvard Ocean Prediction System

IEEE Institute of Electrical e Electronics Engineers

ILU Incomplete Lower Upper Decomposition

IPCC Intergovernmental Panel on Climate Change

MAD Multiply-Add

MICOM Miami Isopycnic Coordinate Ocean Model

MIMD Multiple Instruction Multiple Data

MINRES Minimal Residual

MIT Massachusetts Institute of Technology

MITGCM MIT Global Circulation Model

MOM Modular Ocean Model

MPI Message Passing Interface

MT Multi Threaded instruction fetch e Issue unit

NCAR National Center Atmospheric Research

NCOM NCAR Community Ocean Model

NEMO Nucleus for European Modeling of the Ocean

NLOM Navy Layered Ocean Model

OGCM Ocean Global Circulation Model

OPA Ocean Parallelise

PCG Preconditioned Conjugate Gradient

POM Princeton Ocean Model

POP Parallel Ocean Model

ROMS Regional Ocean Modeling System

ROP Raster Operation Processor

ROP Texture Processor Cluster

SAINV Sparse Approximate Inverses

SFU Special Function Unit

SIMD Single Instruction Multiple Data

SIMT Single Instruction Multiple Threads

SM Stream Multiprocessor

SMC Streaming Multiprocessor Controller

SOR Successive Over Relaxation

SP Streaming Processor

SPA Scalable Processor Array

TOMS (Terrain Following Ocean Modeling System)

TPC Texture Processor Cluster

Indice

1 I Modelli Numerici Oceanici **14**

1.1 Introduzione 14

1.2 I Modelli Numerici Oceanici Globali 19

1.2.1 Rassegna dei Principali Modelli 20

1.2.2 Il Modello Matematico 22

1.2.3 Condizioni Cinematiche al Contorno 26

1.2.4 Metodi di Approssimazione Spazio-Temporali 28

1.3 La Risoluzione Numerica 35

1.3.1 Esempio di Discretizzazione 35

1.3.2 La Classe dei Metodi Iterativi 41

1.3.3 Solutori per Matrici Simmetriche 45

2 Precondizionamento basato sull'Inversione della Decomposizione di Cholesky in un Modello Oceanico Globale **48**

2.1 Introduzione 48

2.2 Il Nucleo Ellittico del Modello Oceanico 51

2.3 Preliminari sul Modello Numerico 54

2.4 Precondizionamento e Convergenza nel Modello Oceanico . . . 59
2.5 Un Precondizionatore Inverso Sparso nel Modello Oceanico . . 64
2.6 Esperimenti Numerici 75
2.7 Conclusioni 81

3 Il GPU Computing nella Risoluzione del Nucleo Ellittico di un Modello Oceanico 83

3.1 Introduzione 83
3.2 Approfondimenti sul Nucleo Ellittico in OPA-NEMO 85
3.3 Risoluzione del Problema mediante PCG su GPU 91
3.4 Precondizionatori Inversi nel Modello Oceanico su GPU 97
3.5 Esperimenti numerici 100
3.6 Conclusioni 111

A Appendice 112

A.1 L'Architettura NVIDIA Tesla 112
A.2 Schema delle Operazioni su GPU 113
A.3 Le Componenti dello SPA 114
A.4 Un' architettura di tipo SIMT 117
A.5 Aritmetica Floating Point nelle GPU 119
A.6 Il Modello di Programmazione CUDA 120
A.7 Gerarchia dei Thread 121
A.8 Configurazione di Esecuzione del Kernel 124
A.9 Gerarchia della Memoria 125
A.10 Sincronizzazione 128

A.11 Uso della Piattaforma 129

A.12 Il Modello di Programmazione CUBLAS 130

Capitolo 1

I Modelli Numerici Oceanici

1.1 Introduzione

Le simulazioni computazionali per la previsione della circolazione degli oceani si ottengono per mezzo delle soluzioni numeriche delle equazioni differenziali alle derivate parziali della dinamica dei fluidi. L'equazioni sono applicate, nello specifico, a sottili strati di fluidi, il cui moto, avviene su una sfera rotante [62, 91].

L' atmosfera e gli oceani muovono grandi quantità di calore dalle regioni tropicali alle regioni alle più alte latitudini. Questo trasporto modera le differenze di temperatura causate dal diverso riscaldamento solare. Nel caso specifico degli oceani, la maggior parte del trasporto di calore è dovuto alle intense correnti presenti lungo i bordi occidentali dei bacini oceanici e alla turbolenza marina provocata da fenomeni di vorticità.

Un'importante componente del flusso oceanico è la circolazione termoalina,

dove le acque calde corrono dall' oceano Indiano fino al lontano nord Atlantico come mostrato in figura (1.1), diventando sempre più fredde e più salate, a causa della presenza del ghiaccio marino. Queste acque, a causa della crescente densità, affondano muovendosi cosí nuovamente verso sud lungo i fondali marini per divenire parte di una circolazione globale oceanica con un periodo che è approssimativamente di circa un millennio. Per questi motivi la circolazione globale degli oceani ha un ruolo predominante nel sistema climatico globale.

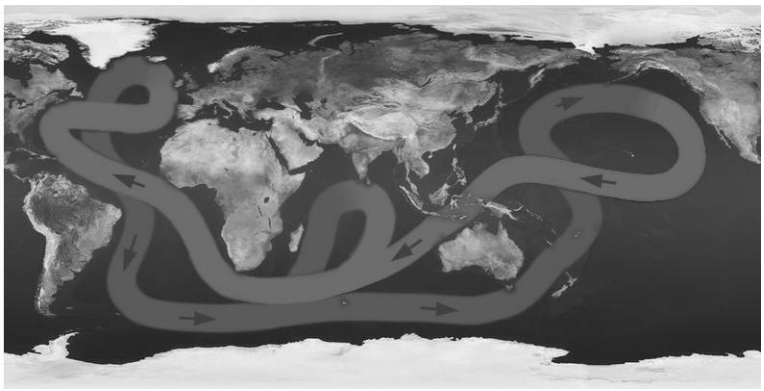

Figura 1.1: La circolazione globale termoalina degli oceani

Una rappresentazione più dettagliata della circolazione degli oceani, ad un fissato istante temporale, è data in figura (1.3), ottenuta per mezzo di una simulazione numerica usando il modello oceanico globale Miami Isopycnic Coordinate Ocean Model (MICOM) [35]. Nella figura le regioni in nero rappresentano le masse terrestri, mentre le varie gradazioni di grigio indicano le temperature delle acque in movimento. Le acque più calde sono posizionate più a sud, queste si muovono per mezzo delle correnti marine nella direzione nord lungo la costa orientale degli Stai Uniti, per poi attraversare la parte

interna del nord Atlantico. Si può osservare sempre dalla figura (1.3) come le correnti oceaniche siano composte da meandri di piccola scala e vorticità in aggiunta al loro movimento di più grande scala.

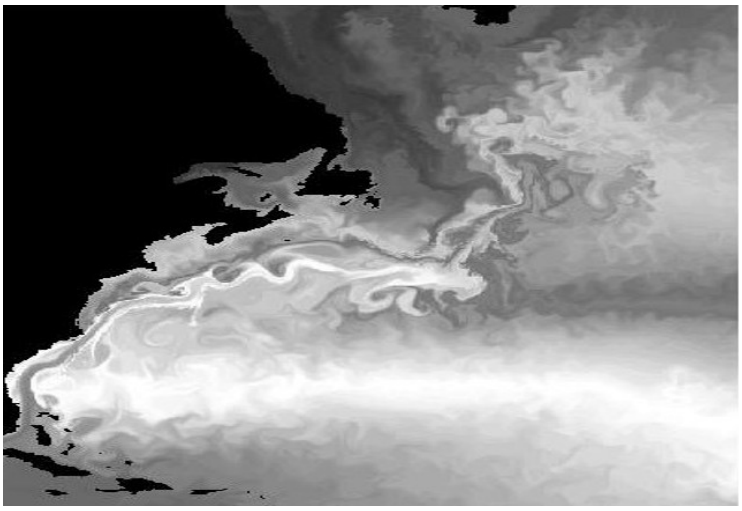

Figura 1.2: Temperatura superficiale del mare nell'Oceano Atlantico occidentale ottenuto da una simulazione numerica fornita dal Rainer Bleck NASA Goddard Institute for Space Studies

Al fine di valutare i possibili scenari futuri sull' evoluzione del clima sulla Terra, numerosi gruppi hanno usato simulazioni numeriche basate su modelli climatici accoppiati, i quali includono moduli per la previsione atmosferica, degli oceani, dei mari glaciali e degli effetti della superficie terrestre. Questi modelli sono eseguiti simulando centinaia di anni di evoluzione climatica, di conseguenza sono computazionalmente molto onerosi. Estese informazioni sull'uso di questi modelli sono incluse nei rapporti tecnici dell' Intergovernmental Panel on Climate Change (IPCC).

Inoltre aspetti di più piccola scala della circolazione oceanica sono anche di

interesse sociale e scientifico. Per esempio, in certe regioni costiere come lungo il nord-ovest degli Stati Uniti, i venti di più forte intensità spesso causano lo spostamento delle acque superficiali in mare aperto, in modo tale che le acque più fredde in profondità salgano in superficie portando con se nutrienti essenziali per la catena alimentare marina [48]. Queste multiple ragioni spingono ad ottenere sempre più dettagliata comprensione della circolazione oceanica.

La simulazione numerica è uno strumento importante per ottenere questo fine. La progettazione di algoritmi numerici per questo scopo è fortemente influenzata dalle proprietà fisiche dei flussi oceanici e dalle proprietà matematiche delle soluzioni dell' equazioni alla base dei modelli oceanici [62].

Le equazioni matematiche sulla quale questi modelli si basano sono le note equazioni primitive.

$$\frac{\partial \mathbf{U}_h}{\partial t} = -\left[(\nabla \times \mathbf{U}) \times \mathbf{U} + \frac{1}{2}\nabla(\mathbf{U}^2)\right]_h - f\mathbf{k} \times \mathbf{U}_h - \frac{1}{\rho_0}\nabla_h(p_i + p_s) + \mathbf{D}^{\mathbf{U}} \quad (1.1)$$

$$\frac{\partial p_i}{\partial z} = -\rho g \quad (1.2)$$

$$\nabla \cdot \mathbf{U} = 0 \quad (1.3)$$

$$\frac{\partial T}{\partial t} = -\nabla \cdot (T\mathbf{U}) + D^T \quad (1.4)$$

$$\frac{\partial S}{\partial t} = -\nabla \cdot (S\mathbf{U}) + D^S \quad (1.5)$$

$$\varrho = \varrho(T, S, p) \quad (1.6)$$

L'incognita U rappresenta il campo delle velocità tridimensionali mentre U_h è il campo delle velocità orizzontali . Le variabili T, S, p_i sono rispettivamente la temperatura, la salinità e la pressione idrostatica. Infine D^U D^T

e D^S simboleggiano fenomeni di piccola scala e p_s è la pressione superficiale atmosferica. Le principali differenze di questi modelli oceanici sono [54]:

- nella rappresentazione dei fenomeni di diffusione D^U, D^T e D^S in funzione delle incognite U, T, S e il calcolo del gradiente di pressione superficie $\nabla_h p_s$ al fine di chiudere il modello.

- nella scelta delle coordinate verticali come i z-model, i ϱ-model i σ-model per la rappresentazione della fisica verticale del fluido oceanico e la scelta delle griglie orizzontali, come la A-grid, la B-grid, la C-grid e la E-grid utilizzata per la rappresentazione delle quantità fisiche di interesse del modello dinamico.

- nell'uso di metodi numerici, come espliciti, impliciti o semi impliciti, utilizzati per discretizzare le equazioni evolutive nelle equazioni primitive.

Il modello matematico composto dalle equazioni (3.2), (2.2), (2.4), (1.5), (1.6) cosi come è presentato non è "chiuso" in quanto il numero di equazioni è inferiore al numero delle quantità incognite presenti. Generalmente il tipo di equazione che aggiunge per chiudere questi modelli alle primitive è quasi sempre di tipo ellittico, come per il modello oceanico NEMO [74] , che per calcolare il gradiente di pressione superficiale nell'equazione di Navier Stokes aggiunge appunto un tale problema. Inoltre gli schemi di discretizzazione spazio temporale per i modelli oceanici sono quasi sempre alle differenze finite, le quali conducono a sistemi lineari con matrici incomplete con un

elevato grado di sparsità e particolari proprietà, quali ad esempio la simmetria, la definita positività e simili. Applicare metodi diretti per la risoluzione di questi sistemi, non sfrutterebbe queste proprietà e le conseguenze sarebbero elevati costi computazionali a causa delle grandi risoluzioni delle griglie spazio temporali utilizzate dai modelli in questione. Per sfruttare tali proprietà i metodi di tipo iterativo si prestano molto bene se dotati di un' elevata rapidità di convergenza per la determinazione delle soluzioni.

1.2 I Modelli Numerici Oceanici Globali

E'noto che le soluzioni analitiche delle equazioni del moto (3.2) sono difficile o impossibili da determinare a causa dei flussi tipici dell'Oceano. Il problema principale è dovuto ai termini non-lineari presenti nelle equazioni primitive, all'attrito ed alla necessità di funzioni che rappresentano la forma reale del fondo marino e delle linee di costa. Inoltre è risaputo come sia difficile descrivere l' Oceano dalle sole misure provenienti come dai satelliti, che possono osservare solo alcuni processi su intervalli di pochi giorni sulla Terra e soltanto vicino alla superficie. Ulteriori misurazioni possono provenire anche dalle navi, che possono stimare molti più parametri, ed anche in profondità, ma le misure sono dispersive non riuscendo a ricoprire tutto il globo terrestre. Quindi, i modelli numerici forniscono la sola utile visione globale delle correnti oceaniche a lungo termine.

1.2.1 Rassegna dei Principali Modelli

Al momento nel campo della modellizzazione per la circolazione oceanica tre sono le classi differenti di modelli numerici oceanici largamente utilizzati da parte delle diverse comunità scientifico climatiche. Queste collettività apportano continuamente a questi modelli innovazioni di tipo fisico e numerico e favoriscono la diffusione del relativo software e documentazione attraverso il World Wide Web.

Queste classi di modelli sono caratterizzate in base al loro approccio alla discretizzazione spaziale del dominio: differenze finite, elementi finiti o volumi finiti. Inoltre all'interno di ogni classe c'è un ulteriore classificazione secondo il tipo di trattamento delle coordinata verticale impiegata per cui si classificano i $z-$models, $\varrho-$ models e $\sigma-$models [54].

La prima classe di modelli oceanici chiamati Ocean Global Circulation Model (OGCM), attualmente la più diffusa fu introdotta da Kirk Bryan e i suoi colleghi del Geophysical Fluid Dynamics Laboratory (GFDL), i quali utilizzarono le differenze finite per le approssimazioni delle equazioni primitive scritte nelle coordinate geopotenziali ($z-$models) [54]. Al momento, variazioni di questa classe di modelli sono realizzate ad Harvard con il modello Harvard Ocean Prediction System (HOPS) [89], al GFDL per mezzo del modello Modular Ocean Model (MOM) [56], al Los Alamos National Laboratory mediante il modello Parallel Ocean Model (POP) [66], al National Center Atmospheric Research (NCAR) attraverso il modello NCAR Community Ocean Model (NCOM) [94] ed infine al Lodyc Institute con il modello Nucleus for European Modeling of the Ocean (NEMO) [74]. Alcuni di questi

modelli alle equazioni primitive scritte nelle coordinate geopotenziali sono stati in seguito discretizzati anche mediante i volumi finiti, come ad esempio, al Massachusetts Institute of Technology (MIT) con il modello MIT Global Circulation Model (MITGCM) [2].

La descrizione della fisica verticale degli oceani attraverso un opportuno sistema di riferimento è un aspetto fondamentale nei modelli oceanici. Durante gli anni 70, oltre al sistema geopotenziale, comparirono due nuovi approcci per tale proposito. I modelli che utilizzavano questi metodi novelli erano basati rispettivamente sulla rappresentazione degli strati immiscibili oceanici ("layered models" o "$\varrho-$models ") e quelli basati sulla mappatura del fondo marino ("terrain following models" o $\sigma-$models") [54]. Con la linea di pensiero di quegli anni entrambi le classi utilizzavano ancora le differenze finite, come per i modelli geopotenziali, per l' approssimazione delle equazioni primitive. Ancora oggi, diversi modelli di tipo "layered" o "terrain following" vengono ancora impiegati per le simulazioni globali a lungo termine. Tanto per citarne qualcuno, alla prima categoria fanno parte, ad esempio, modelli progettati e utilizzati presso il Naval Research Laboratory con il Navy Layered Ocean Model (NLOM) [101] , all'Università di Miami con il MICOM [71], al GFDL con il Hallberg isopicnica Model (HIM) [59], e altri. Mentre alcuni modelli della seconda categoria vengono ideati e usati a Princeton con il Princeton Ocean Model (POM) [77], e all' università di Rutgers University e UCLA con il modello Regional Ocean Modeling System (ROMS) [85, 93]. I più recenti OGCM sono stati realizzati con l'ausilio di più avanzate, ma meno tradizionali, tecniche numeriche. I più importanti modelli sono stati

sviluppati utilizzando schemi numerici agli elementi finiti, come ad esempio, il modello QUODDI [99] dell' Università di Dartmounth e agli elementi spettrali finiti, come ad esempio, SEOMS [64] dell' Università di Rutger. Questi differiscono fondamentalmente dai precedenti modelli per gli algoritmi utilizzati per risolvere le equazioni del moto e per il loro uso di griglie orizzontali non strutturate [84]. Recentemente anche i metodi ai volumi finiti sono stati utilizzati nei modelli oceanici, come ad esempio il modello FVCOM [25] per l'utilizzo delle griglie non strutturate.

1.2.2 Il Modello Matematico

Tutti i modelli oceanici si basano sul modello matematico delle equazioni primitive [62]. Le equazioni primitive scritte nelle coordinate geopotenziali ($z-$models) sono, come già detto, la base dei più diffusi OGCM fino ad oggi. Tuttavia in alcune applicazioni può essere conveniente applicare delle opportune trasformazioni della coordinata verticale. I cambiamenti di coordinate più adoperati sono i $\sigma-$coordinate e i $\varrho-$coordinate. Il primo trasporta la totale profondità oceanica nell'intervallo $[0, 1]$, associando il valore 0 al fondo oceanico e il valore 1 alla superficie degli oceani, con potenziali vantaggi nel rappresentazione dei processi bentonici. Mentre nel secondo ogni livello verticale è rappresentato da uno strato di fluido di costante densità con conseguenti vantaggi nella rappresentazione della circolazione termoalina [62]. In un sistema generalizzato, le equazioni primitive sono date dalle seguen-

ti [17]:

$$\frac{\partial}{\partial t}\left(u\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial x}\left(uu\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial y}\left(vu\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial Z}\left(\frac{\partial Z}{\partial t}u\frac{\partial p}{\partial Z}\right)+$$
$$-fv\frac{\partial p}{\partial Z}=g\left(\frac{\partial p}{\partial x}\frac{\partial z}{\partial Z}-\frac{\partial p}{\partial Z}\frac{\partial z}{\partial x}\right) \tag{1.7}$$

$$\frac{\partial}{\partial t}\left(v\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial x}\left(uv\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial y}\left(vv\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial Z}\left(\frac{\partial Z}{\partial t}v\frac{\partial p}{\partial Z}\right)+$$
$$+fu\frac{\partial p}{\partial Z}=g\left(\frac{\partial p}{\partial y}\frac{\partial z}{\partial Z}-\frac{\partial p}{\partial Z}\frac{\partial z}{\partial y}\right) \tag{1.8}$$

$$\frac{\partial p}{\partial Z}=-g\varrho\frac{\partial z}{\partial Z} \tag{1.9}$$

$$\frac{\partial}{\partial t}\left(T\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial x}\left(uT\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial y}\left(vT\frac{\partial p}{\partial Z}\right)+\frac{\partial Z}{\partial t}T\frac{\partial p}{\partial Z}=0 \tag{1.10}$$

$$\frac{\partial}{\partial t}\left(S\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial x}\left(uS\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial y}\left(vS\frac{\partial p}{\partial Z}\right)+\frac{\partial Z}{\partial t}S\frac{\partial p}{\partial Z}=0 \tag{1.11}$$

$$\frac{\partial}{\partial t}\left(\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial x}\left(u\frac{\partial p}{\partial Z}\right)+\frac{\partial}{\partial y}\left(v\frac{\partial p}{\partial Z}\right)+\frac{\partial Z}{\partial t}\frac{\partial p}{\partial Z}=0 \tag{1.12}$$

$$\tag{1.13}$$

Nelle equazioni (1.7), (1.8), (1.9), (1.10), (1.11) e (1.12) la coppia (u,v) rappresenta il campo delle velocità orizzontali, p è la somma della pressione idrostatica e superficiale e infine T,S e ϱ sono rispettivamente la temperatura, la salinità e la densità del fluido oceanico. Nelle equazioni primitive la coordinata generalizzata verticale Z può assumere il valore di $z-$coordinate $\sigma-$coordinate oppure $\varrho-$coordinate. Nel modello continuo, questi sistemi di coordinamento verticale sono naturalmente tutti equivalenti tra loro . Purtroppo, le approssimazioni che portano ad un modello numerico introducono degli errori di troncamento creando un diverso comportamento del modello numerico quando si passa da un sistema di coordinate verticale all' altro. Quindi, ciascuno di questi sistemi può essere più adatto per certe

classi di problemi che per altri. Negli ultimi anni, una varietà di modelli di circolazione oceanica, con questi diversi sistemi di coordinate verticale, sono stati sviluppati per applicazioni sia per previsioni a scala regionale e sia a scala globale. Alcuni di questi sono citati in un inventario internazionale di modelli, realizzato da Haidvogel e Beckmann [58], che descrive più di una dozzina di codici appartenenti alle diverse classi di modelli che usano i diversi sistemi di coordinate verticali.

La prima implementazione numerica delle equazioni primitive è stata per mezzo del modello MOM, eseguita presso il GFDL da Bryan [22] e Cox [27]. L'obiettivo originale di questo progetto era quello di costruire uno strumento per la modellazione numerica di grande risoluzione per la previsione della circolazione oceanica. Gli algoritmi in quel momento erano basati sulla decomposizione del dominio globale in domini rettangolari di dimensioni variabili, e sull'uso delle differenze finite per le approssimazioni delle equazioni primitive. In impostazione predefinita, le equazioni sono risolte nelle coordinate sferiche (λ, ϱ, z), dove le prime 2 componenti sono note rispettivamente come longitudine e latitudine mentre z è la profondità degli oceani. La formulazione del livello geopotenziale z è un concetto semplice, che è stato impiegato per una gran varietà di modelli come anche in NEMO , che uno dei più utilizzati modelli oceanografici globali in europa al momento. Le applicazioni principali del modello MOM come per NEMO, POP et all. sono state le simulazioni del comportamento della circolazione termoalina degli oceani sul dominio globale terrestre (figura (1.1)), integrando le equazioni primitive su scale temporali che vanno da anni a secoli.

Nelle coordinate geopotenziali $Z = z$, $\frac{\partial Z}{\partial t}$ coincide con la velocità verticale w del fluido oceanico e l'equazione del moto verticale diventa la nota equazione di idrostatica:

$$\frac{\partial p}{\partial z} = -g\varrho. \tag{1.14}$$

Le equazioni primitive nelle coordinate sferiche e nel sistema geopotenziale, utilizzate nei modelli oceanici come MOM, POP e NEMO et all. [54], diventano:

$$\frac{\partial u}{\partial t} + \frac{1}{r_E \cos\phi}\frac{\partial}{\partial \lambda}(uu) + \frac{1}{r_E \cos\phi}\frac{\partial}{\partial \phi}(v\cos\phi u) + \frac{\partial}{\partial z}(wu) - \frac{uv}{r_E}tan\phi$$
$$-fv = -\frac{1}{r_E \cos\phi}\frac{\partial P}{\partial \lambda} + D^u + F^u \tag{1.15}$$

$$\frac{\partial v}{\partial t} + \frac{1}{r_E \cos\phi}\frac{\partial}{\partial \lambda}(uv) + \frac{1}{r_E \cos\phi}\frac{\partial}{\partial \phi}(v\cos\phi v) + \frac{\partial}{\partial v}(wu) - \frac{u^2}{r_E}tan\phi$$
$$+fv = -\frac{1}{r_E \cos\phi}\frac{\partial P}{\partial \lambda} + D^v + F^v \tag{1.16}$$

$$\frac{\partial P}{\partial z} = -\frac{g}{\varrho_0}\varrho \tag{1.17}$$

$$\frac{\partial T}{\partial t} + \frac{1}{r_E \cos\phi}\frac{\partial}{\partial \lambda}(uT) + \frac{1}{r_E \cos\phi}\frac{\partial}{\partial \phi}(v\cos\phi T) + \frac{\partial}{\partial z}(wT)$$
$$= D^T + F^T \tag{1.18}$$

$$\frac{\partial S}{\partial t} + \frac{1}{r_E \cos\phi}\frac{\partial}{\partial \lambda}(uS) + \frac{1}{r_E \cos\phi}\frac{\partial}{\partial \phi}(v\cos\phi S) + \frac{\partial}{\partial z}(wS)$$
$$= D^S + F^S \tag{1.19}$$

$$\frac{1}{r_E \cos\phi}\left(\frac{\partial u}{\partial \lambda} + \frac{\partial u}{\partial \phi}(v\cos\phi)\right) + \frac{\partial w}{\partial z} \tag{1.20}$$

$$\varrho = \varrho(T, S, z) \tag{1.21}$$

dove $P = \frac{p}{\varrho_0}$ è la pressione dinamica, mentre D e F denotano rispettivamente termini di dissipazione e forzanti al sistema dinamico in esame.

Infine l'equazione di stato (1.21) è di tipo non lineare ed è implementata per ogni livello del modello separatamente mediante un' approssimazione polinomiale del terzo grado dell'equazione di stato degli oceani [23].

1.2.3 Condizioni Cinematiche al Contorno

Nei modelli oceanici, le condizioni cinematiche poste sul bordo verticale superficiale del dominio Ω di simulazione sono le seguenti:

$$w(\lambda, \phi, t) \begin{cases} 0 & "rigid\ lid\ approximation" \\ \dfrac{\partial \eta}{\partial t} & "free\ sea\ surface". \end{cases} \tag{1.22}$$

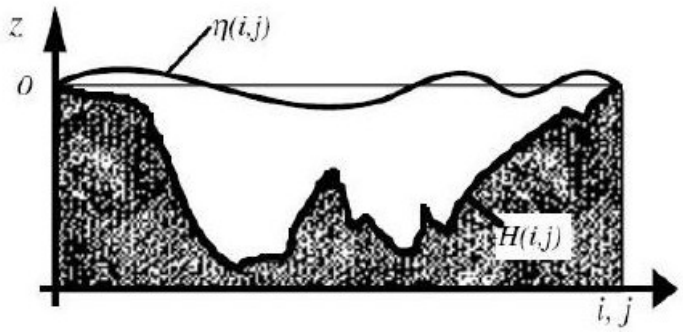

Figura 1.3: Profilo della profondità marina

Il primo vincolo in (1.22) è assunto se si suppone che la superficie degli oceani sia rigida "rigid lid approximation' [62] mentre la seconda condizione "free sea surface" [62] è posta se si assume la superficie degli oceani libera dove la funzione η rappresenta l'inalzamento della superficie marina da una posizione

di riferimento solitamente posta a $z = 0$ come mostrato in figura 1.3. Mentre sul fondo marino solitamente la condizione cinematica al bordo è la seguente:

$$w = -\frac{u}{\cos\phi}\frac{\partial h}{\partial \lambda} - v\frac{\partial h}{\partial \phi} \tag{1.23}$$

dove H è la superficie che rappresenta il fondo oceanico. La condizione in (1.23) esterna una condizione di perpendicolarità tra la normale alla superficie H e con il vettore velocità $\mathbf{U}$. La maggior parte dei modelli oceanici per la parte verticale utilizzano la"rigid lid approximation", dove il flusso verticale delle velocità integrato lungo la profondità è posto uguale a zero, e la relativa streamfunction è calcolata mediante la risoluzione di un problema ellittico [74] . La "mass streamfunction trasport" ψ è definita nel seguente modo:

$$\begin{aligned} U &= -\frac{1}{hr_E}\frac{\partial \psi}{\partial \phi} \\ V &= -\frac{1}{hr_E}\frac{\partial \psi}{\partial \lambda} \end{aligned} \tag{1.24}$$

dove U e V sono le velocità integrate lungo le profondità oceaniche e r_E è il raggio medio terrestre. La funzione ψ è variabile nel tempo ed è determinata da un equazione ottenuta calcolando il rotore delle equazioni del moto integrate lungo la profondita degli oceani. L'equazione di vorticità risultante è dipendente dal tempo ed è unita a delle opportune condizioni al bordo [74]. Le velocità medie vengono calcolate dopo aver risolto il problema ellittico con l'ausilio delle equazioni in (1.24). Recentemente, due strategie aggiuntive per la parte superficiale della circolazione oceanica sono state implementate; uno si basa su una superficie libero implicita [38] degli oceani, l'altra sulla risoluzione di un' equazione ellittica nell'incognita P, al posto della funzione ψ [39]. Inoltre un modello oceanico con la superficie libera es-

plicita η al posto di uno con una superficie libera implicita ha un elevato time splitting [69]. Queste due nuove strategie hanno vantaggi per l'inclusione delle isole nel dominio di integrazione e sono potenzialmente più efficienti in applicazioni ad alta risoluzione.

Infine le condizioni al contorno laterale sulla velocità sono ancora date da una condizione di ortogonalità:

$$\mathbf{U} \cdot \nu = 0 \tag{1.25}$$

dove $\mathbf{U}$ è il vettore velocità mentre ν è la normale alla superficie che rappresenta il bordo laterale.

1.2.4 Metodi di Approssimazione Spazio-Temporali

Per quanto riguarda la discretizzazione spaziale, i più diffusi modelli oceanici utilizzano una discretizzazione basata sul cosiddetto "box-concept", in cui il dominio di simulazione è suddiviso in una serie di celle rettangolari (A-grid, B-grid, C-grid e E-grid) [3], in cui le variabili fisiche di interesse sono calcolate su parti differenti di queste celle. La figura (1.4) mostra due esempi di discretizzazione in un'area con forti variazioni della profondità del mare. Il primo esempio utilizza una discretizzazione con livelli allo stesso geopotenziale equidistanti, mentre la seconda mostra una discretizzazione con livelli ancora allo stesso geopotenziale ma sempre più vicini in prossimità della superficie marina. La seconda rappresentazione a differenza della prima migliora la descrizione dei processi termoalini ed è tipica della maggior parte dei modelli oceanici [26, 21, 47, 92, 72].

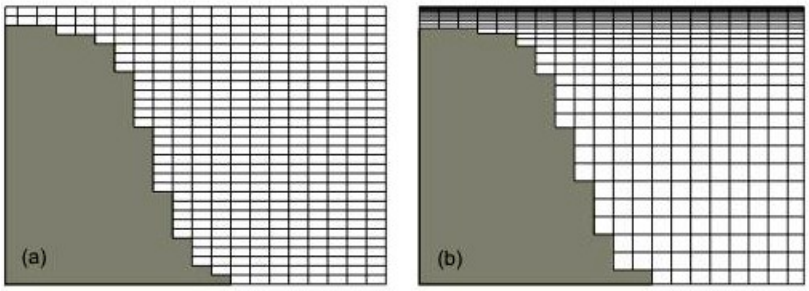

Figura 1.4: Discretizzazione di un profilo verticale marino, con 30 livelli a spaziatura equidistante della griglia, nel sistema di coordinate geopotenziale (a), Discretizzazione della stessa topografia con più alta risoluzione in prossimità della superficie marina (b).

Si noti in figura (1.4), in aggiunta, che la forma della topografia del profilo marino cambia a causa della risoluzione verticale utilizzata ed dal posizionamento delle celle della griglia, soprattutto nelle regioni con più forti pendenze rispetto a quelle più regolari. Questo causa due problemi distinti per la scelta dei livelli: la rappresentazione della topografia e la rappresentazione della stratificazione. Esiste tuttavia una certa flessibilità nella scelta dei livelli, sebbene sia possibile l'uso di una griglia la cui risoluzione varia in funzione del gradiente di una funzione regolare analitica che rappresenta il profilo marino in questione [98]. Questo permette di trovare una discretizzazione adatta sia per la topografia del dominio che per la stratificazione del fluido oceanico. Lo svantaggio della discretizzazione verticale del sistema di coordinate geopotenziale mostrata in figura (1.4) è che essa può portare a un gran numero di celle della griglia inattive, che aumentano sia i costi della memoria dedicata del sistema su cui si lavora e sia il numero di operazioni

non utili ai fini della predizione oceanografica. Si noti inoltre che questa scelta di discretizzazione è dovuta alla condizione (1.25). Una simile discretizzazione porta ad un errore numerico sistematico associato alla perdita di energia baroclinica nel modello. In generale, al fine di eliminare questo problema, per la rappresentazione ottimale della topografia si fa riferimento a [102].

Per le approssimazioni delle equazioni primitive in coordinate sferiche (1.15), (1.16),(1.17),(1.18) (1.19) e (1.20), le tecniche di discretizzazioni più diffuse, come detto in precedenza, sono le differenze finite.

Data una funzione regolare $f(\lambda, \phi, z)$, si definisce i seguenti operatori media nelle tre direzioni corrispondenti alla base scelta per la rappresentazione del modello:

$$\begin{aligned} \bar{f}^{\lambda} &= \frac{1}{2}(f_{i+1,j,k} + f_{i,j,k}) \\ \bar{f}^{\phi} &= \frac{1}{2}(f_{i,j+1,k} + f_{i,j,k}) \\ \bar{f}^{z} &= \frac{1}{2}(f_{i,j,k+1} + f_{i,j,k}) \end{aligned} \tag{1.26}$$

e i seguenti operatori discreti della derivata:

$$\begin{aligned} \delta_{\lambda} f &= \frac{1}{r_E \Delta\lambda}(f_{i+1,j,k} - f_{i,j,k}) \\ \delta_{\phi} f &= \frac{1}{r_E \Delta\phi}(f_{i,j+1,k} - f_{i,j,k}) \\ \delta_{z} f &= -\frac{1}{r_E \Delta z}(f_{i,j,k+1} - f_{i,j,k}) \end{aligned} \tag{1.27}$$

Il segno negativo nell'ultima relazione riflette il fatto che l'incremento dell' indice k avviene in senso opposto alla verso crescente di z. Utilizzando gli operatori discreti le equazioni primitive in un forma semi discreta rispetto

allo spazio diventano:

$$\frac{\partial u}{\partial t}+\delta_\lambda\left(\frac{1}{\overline{\overline{\Delta y}\phi\lambda}}\bar{u}^\lambda\overline{\Delta y}^\phi\bar{u}^\lambda\right)+\delta_\phi\left(\frac{1}{\overline{\overline{\Delta x}\phi\lambda}}\bar{v}^\lambda\overline{\Delta x}^\lambda\bar{u}^\phi\right)+\delta_z(\bar{w}^{\phi\lambda}\bar{u}^z)-fv= \\ -\frac{1}{\cos\bar{\phi}^\phi}\delta_\lambda(\bar{P}^\phi)+D^u+F^u \quad (1.28)$$

$$\frac{\partial v}{\partial t}+\delta_\lambda\left(\frac{1}{\overline{\overline{\Delta y}\phi\lambda}}\bar{u}^\lambda\overline{\Delta y}^\phi\bar{v}^\lambda\right)+\delta_\phi\left(\frac{1}{\overline{\Delta x}^{\phi\lambda}}\bar{v}^\phi\overline{\Delta x}^\lambda\bar{v}^\phi\right)+\delta_z(\bar{w}^{\phi\lambda}\bar{v}^z)-fu= \\ +\delta_\phi(\bar{P}^\lambda)+D^v+F^v \quad (1.29)$$

$$P=\frac{g}{\varrho_0}\sum_0^z(\bar{\varrho}^z)\overline{\Delta z}^z \quad (1.30)$$

$$\frac{\partial T}{\partial t}+\delta_\lambda\left(\frac{1}{\Delta y}\overline{u(\overline{\Delta y}^\phi)^\phi\bar{T}^\lambda}\right)+\delta_\phi\left(\frac{1}{\Delta x}\overline{u(\overline{\Delta x}^\lambda)^\lambda\bar{T}^\phi}\right)+\delta_z(w\bar{T}^z) \quad (1.31)$$

$$=D^T+F^T \quad (1.32)$$

$$\frac{\partial S}{\partial t}+\delta_\lambda\left(\frac{1}{\Delta y}\overline{u(\overline{\Delta y}^\phi)^\phi\bar{S}^\lambda}\right)+\delta_\phi\left(\frac{1}{\Delta x}\overline{u(\overline{\Delta x}^\lambda)^\lambda\bar{S}^\phi}\right)+\delta_z(w\bar{S}^z) \quad (1.33)$$

$$=D^S+F^S \quad (1.34)$$

Dall'equazione di continuità $\nabla\cdot\mathbf{U}=0$ applicata ad un fluido incomprimibile si ricava la velocità verticale:

$$w=\frac{1}{cos\phi}\sum_0^z\left(\delta_\lambda\left(\frac{1}{\Delta y}\overline{(u\overline{\Delta y}^\phi)^\phi}\right)+\delta_\phi\left(\frac{1}{\Delta y}\overline{(v\overline{\Delta y}^\lambda)}^\lambda\right)\Delta z\right) \quad (1.35)$$

Infine la vorticità barotropica è:

$$\zeta=\sum_1^h\left(\Delta x\delta_\lambda(\bar{dy}^\lambda\bar{v}^\phi)-\Delta y\delta_\phi\left(\overline{dx}^\phi\cos\bar{\phi}^\phi\bar{u}^\lambda\right)\right)\Delta z \quad (1.36)$$

mentre le velocità barotropiche in funzione della streamfunction ψ sono:

$$\begin{aligned}U&=-\frac{1}{min_{\lambda\phi}(H)}\delta_\phi\overline{\psi}^\lambda\\ V&=-\frac{1}{min_{\lambda\phi}(H)\cos\bar{\phi}^\phi}\delta_\lambda\overline{\psi}^\lambda\end{aligned} \quad (1.37)$$

dove $min_{\lambda\phi}(H)$ è il minimo assoluto della funzione H che parametrizza il fondo.

Le tecniche di discretizzazione utilizzate nei modelli oceanici per le approssimazioni temporali dei processi non diffusivi all'interno delle equazioni primitive sono gli schemi centrati di *Leap Frog* a tre livelli [3], dove indicato con u la funzione incognita, si ha:

$$u^{t+\Delta t} = u^{t-\Delta t} + 2\Delta t RHS^t \tag{1.38}$$

dove RHS é la funzione differenziale dell'equazione data, Δt è il passo di discretizzazione e gli indici $t + \Delta$, $t - \Delta t$ e t sono i tempi nella quale le quantità presenti nella (1.38) vengono valutate. Il primo step dello schema (1.38) quando si parte da una condizione a riposo dell'oceano è dato dal seguente:

$$u^1 = u^0 + 2\Delta t RHS^0 \tag{1.39}$$

Questo schema é ampiamente diffuso per i processi di avvezione per i fluidi a bassa viscositá. Lo schema di *Leap Frog* a tre livelli è un metodo efficiente che ha un'accuratezza nel tempo del secondo ordine con solo la valutazione della parte destra di (1.38) per ogni passo temporale. Inoltre non smorza i moti oscillatori lineari ne produce instabilità amplificando le oscillazioni. Questi vantaggi sono spesso diminuiti dalla grande velocità di fase dell'errore, caratteristica degli schemi *Leap Frog* e dalla poca adeguatezza per le approssimazioni dei processi diffusivi e dello smorzamento di Rayleight. Tut-

tavia il problema più grande associato a questi schemi di discretizzazione è l'alta frequenza del rumore computazionale chiamato *time splitting* [60] che si genera quando questi vengono applicati a modelli non lineari risolti mediante opportune tecniche numeriche come in [28]. il fenomeno del *time splitting* viene controllato usualmente attraverso l'uso del filtro di Asselin [88, 4] dato da:

$$u_f^t = u^t + \gamma\left(u_f^{t-\Delta t} - 2u^t + u^{t+\Delta t}\right) \tag{1.40}$$

dove u_f^t e $u_f^{t-\Delta t}$ sono i valori della soluzione u ottenuti dal filtro al tempo t e $t - \Delta t$ e γ è il coefficiente di Asselin. L'effetto regolarizzante che ha questo filtro é mostrato in figura (1.5).

Un' altra strategia spesso impiegata è la reinizializzare periodica della

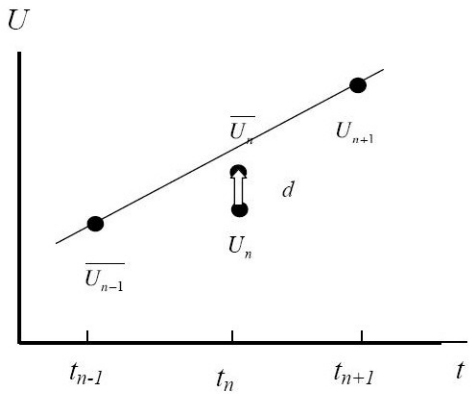

Figura 1.5: Nell'immagine é mostrato l'effetto regolarizzante che il filtro induce sulla soluzione u calcolata al tempo t.

soluzione u mediante una singola integrazione con lo schema *Leap Frog* a

due livelli. Purtroppo entrambe le strategie degradano l'accuratezza del calcolo dal secondo ordine al primo.

Invece per i termini di diffusione orizzontale D nelle equazioni per il calcolo dei traccianti si utilizza solitamente uno schema alle differenze in avanti rappresentato da:

$$u^{t+\Delta t} = u^{t-\Delta t} + 2\Delta t D^{t-\Delta t} \tag{1.41}$$

La condizione di stabilità per lo schema (1.41) è dato dalla seguente relazione tra il passo spaziale e il passo temporale:

$$A^h \leq \frac{\Delta x^2}{\pi^2 2\Delta t} \tag{1.42}$$

con Δx il passo spaziale, Δt il passo temporale e A^h il coefficiente di diffusione orizzontale.

Per i fenomeni di diffusione verticale ci si può servire di uno schema alle differenze in avanti che però può condurre a delle instabilità numeriche le quali implicano un forte vincolo sul passo temporale. Le soluzioni più comuni a questo problema sono o l'uso di uno schema alle differenze in avanti con una tecnica di *time splitting* oppure sostituire lo schema in avanti con uno alle differenze all'indietro.

Nel primo caso il passo temporale Δt è suddiviso in un numero di parti N, in modo tale da cercare di ridurre l'instabilità associata di N volte. Il time splitting è mostrato nel seguente:

$$u^{t-\Delta t+2L\frac{\Delta t}{N}} = u^{t-\Delta t+2(L-1)\frac{\Delta t}{N}} + \frac{2\Delta t}{N} D^{t-\Delta t+2(L-1)\frac{\Delta t}{N}} \quad 1 \leq L \leq N \tag{1.43}$$

Lo schema alle differenze all'indietro è sempre stabile ed è rappresentato

dalla seguente procedimento iterativo:

$$u^{t+\Delta t} = u^{t-\Delta t} + 2\Delta t D^{t+\Delta t}. \tag{1.44}$$

Diversamente da uno schema alle differenze in avanti, che per ogni iterazione, richiede una valutazione della parte destra dell'equazione, lo schema alle differenze all'indietro richiede, la risoluzione di un sistema lineare.

1.3 La Risoluzione Numerica

In questo paragrafo si vuole mostrare come l'applicazione delle differenze finite per l'approssimazione delle equazioni primitive, costituenti i modelli oceani, conducono quasi sempre a sistemi lineari, con particolari proprietà quali: la sparsità, la simmetria e la definita positività. Inoltre si farà vedere come questi sistemi possano essere risolti in maniera efficiente, sfruttando le proprietà appena elencate, da metodi di risoluzione numerica di tipo iterativo in contrapposizione ai classici metodi di risoluzione diretta per sistemi lineari.

1.3.1 Esempio di Discretizzazione

Si considereranno in questo sottoparagrafo il caso di un' equazione di tipo parabolico, che descrive nello specifico dei modelli di circolazione globale, la diffusione del calore e della salinita del fluido oceanico e la sua versione stazionaria ovvero un' equazione di tipo ellittico che nei modelli oceanici è anche essa presente, come accennato nel paragrafo precedente, per il calcolo della streamfunction ψ utile ai fini della determinazione delle velocità barocliniche.

Più in generale, in natura, un numero differente di processi fisici può essere descritto per mezzo dell' equazione di diffusione:

$$\frac{\partial T}{\partial t} - \nabla \cdot (a \nabla T) = f \quad in\ \Omega \tag{1.45}$$

La funzione T rappresenta la distribuzione della temperatura (oppure della salinità) nel fluido oceanico ad un istante t sul dominio di interesse Ω, sulla quale agisce una sorgente esterna di calore f come ad esempio l'irraggiamento solare. La funzione $a(x, y)$, assunta sempre positiva su Ω, in (1.45) è il termine che rappresenta la conduttività termica del mezzo dove il calore si diffonde oppure la velocità con la quale quest'ultimo si muove. Per determinare la temperatura ad un istante t in un punto $(x, y) \in \Omega$ è necessario conoscere la temperatura iniziale $T(x, 0)$ su Ω e le condizioni al bordo di Ω con delle relazioni del tipo:

$$\begin{cases} T(x,0) = h(x) \quad su \quad \Omega. \\ \\ T(x,t) = g(x,t) \quad su \quad \delta\Omega. \end{cases} \tag{1.46}$$

Qui si è posto, il caso particolare della funzione $g(x, t) = 0$ su $\delta\Omega$. Un metodo di uso comune nei modelli oceanici come già detto più volte per ottenere un'approssimazione delle equazioni differenziali alle derivate parziali è il metodo delle differenze finite. Con tali tecniche il dominio continuo è sostituito da una griglia finita Ω_{h_x,h_y} di punti Ω e per ogni punto della griglia le derivate in (1.45) sono approssimate da rapporti incrementali. Le approssimazioni saranno sempre migliori al divenire della griglia Ω_{h_x,h_y} più fitta.

Nel particolare la regione Ω è approssimata da una griglia uniforme $\Omega_{h_x,h_y} = \{(x_i\ y_j) : i = 0, ..., n_x + 1,\ j = 0, ..., n_y + 1\}$ con passo di discretizzazione

$h_x = \frac{1}{n_x+1}$ nella direzione x e $h_y = \frac{1}{n_y+1}$ nella direzione y, come mostrato in figura (1.6) (nel caso specifico di $\Omega = [0,1] \times [0,1]$ con $n_x = 3$ e $n_y = 5$). Applicando le differenze finite centrate in (1.45), che approssimano le derivata parziali seconde si ottiene:

$$\left(\frac{\partial}{\partial x} a \frac{\partial T}{\partial x}\right)(x_i, y_j) \approx \frac{a_{i+1/2,j}(T_{i+1,j} - T_{i,j}) - a_{i-1/2,j}(T_{i,j} - T_{i-1,j})}{h_x^2} \quad (1.47)$$

dove $a_{i\pm1/2,j} \equiv a(x_i \pm h_x/2, y_j)$ e $T_{i,j}$ rappresenta un' approssimazione di $T(x_i, y_j)$ nel punto (x_i, y_j). Un' analoga espressione è ottenuta per le derivate parziali nella direzione y:

$$\left(\frac{\partial}{\partial y} a \frac{\partial T}{\partial y}\right)(x_i, y_j) \approx \frac{a_{i,j+1/2}(T_{i,j+1} - T_{i,j}) - a_{i,j-1/2}(T_{i,j} - T_{i,j-1})}{h_y^2} \quad (1.48)$$

dove $a_{i,j\pm1/2} \equiv a(x_i, y_j \pm hy/2)$.

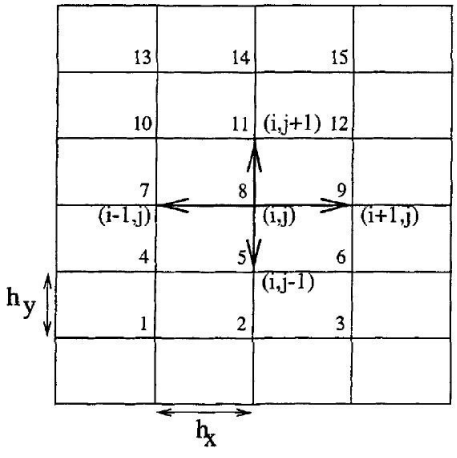

Figura 1.6: Griglia di discretizzazione del dominio per lo schema delle differenze finite

Se si considera il caso stazionario dell'equazione (1.45), ovvero un equazione di tipo ellittico, rappresentato da:

$$\begin{cases} -\nabla \cdot (a \nabla T) = f \quad su\ \Omega \\ \\ T = 0 \quad su\ \delta\Omega. \end{cases} \tag{1.49}$$

utilizzando il metodo delle differenze finite per (1.49) si ricava il problema discreto seguente:

$$\begin{gathered} -\Bigg(\frac{a_{i+1/2,j}(T_{i+1,j} - T_{i,j}) - a_{i-1/2,j}(T_{i,j} - T_{i-1,j})}{h_x^2} + \\ \frac{a_{i,j+1/2}(T_{i,j+1} - T_{i,j}) - a_{i,j-1/2}(T_{i,j} - T_{i,j-1})}{h_y^2} \Bigg) = f_{i,j} \\ i = 1, ..., n_x \quad j = 1, ..., n_y. \end{gathered} \tag{1.50}$$

da cui è possibile ricavare un sistema lineare di $n_x \times n_y$ equazioni algebriche per determinare i valori $T_{i,j}$ all'interno della griglia Ω_{h_x,h_y}

Per il caso non stazionario (1.45), adoperando una differenza finita all'indietro (oppure centrata) per $\dfrac{\partial T}{\partial t}$ ed uno schema temporale in avanti per l'equazione (1.45) si consegue nuovamente ancora ad un problema discreto del tipo:

$$\begin{gathered} \frac{T_{i,j}^{l+1} - T_{i,j}^{l}}{\Delta t} - \Bigg(\frac{a_{i+1/2,j}(T_{i+1,j}^{l+1} - T_{i,j}^{l+1}) - a_{i-1/2,j}(T_{i,j}^{l+1} - T_{i-1,j}^{l+1})}{h_x^2} + \\ \frac{a_{i,j+1/2}(T_{i,j+1}^{l+1} - T_{i,j}^{l+1}) - a_{i,j-1/2}(T_{i,j}^{l+1} - T_{i,j-1}^{l+1})}{h_y^2} \Bigg) = f_{i,j}^{l+1} \\ i = 1, ..., n_x \quad i = 1, ..., n_y \end{gathered} \tag{1.51}$$

da cui si giunge ancora una volta ad un sistema lineare di $n_x \times n_y$ equazioni algebriche. Per scrivere il problema numerico in termini matriciali si deve innanzitutto scegliere un ordinamento per le equazioni e per le incognite $T_{i,j}$ del modello. Una scelta comune è l'ordinamento naturale legato ai punti

della griglia scorrendo da sinistra verso destra e dal basso verso l'alto come mostrato in figura (1.6). Con questo ordinamento del problema (1.50) o del problema (1.51) si giunge ad un sistema di equazioni algebrico lineari indicato con:

$$A\mathbf{T} = \mathbf{f} \tag{1.52}$$

dove A è una matrice a blocchi tridiagonale con n_y blocchi diagonali, ognuno di dimensione n_x^2; $\mathbf{T}$ è il vettore $n_x \times n_y$ contenente i valori di $T_{i,j}$ memorizzati nelle posizioni $(j-1)n_x + i, \quad i = 1, ..., n_x$ e $i = 1, ..., n_y$; e $\mathbf{f}$ è il vettore dei termini noti di dimensione $n_x \times n_y$ contenete i valori di $f_{i,j}$ nella posizione $(j-1)n_x + i, \quad i = 1, ..., n_x$ e $i = 1, ..., n_y$. Se si pone con:

$$\begin{aligned} d_{i,j} &= \frac{a_{i+1/2,j} + a_{i-1/2,j}}{h_x^2} + \frac{a_{i,j+1/2} + a_{i,j-1/2}}{h_x^2} \\ b_{i+1/2,j} &= \frac{-a_{i+1/2,j}}{h_x^2} \quad c_{i,j+1/2} = \frac{-a_{i,j+1/2}}{h_y^2} \end{aligned} \tag{1.53}$$

allora la matrice A può essere scritta nella seguente maniera:

$$A = \begin{pmatrix} S_1 & T_{3/2} & 0 & 0 \\ T_{3/2} & S_2 & \ddots & 0 \\ 0 & \ddots & \ddots & T_{n_y-\frac{1}{2}} \\ 0 & 0 & T_{n_y-\frac{1}{2}} & S_{n_y} \end{pmatrix} \tag{1.54}$$

dove

$$S_j = \begin{pmatrix} d_{1,j} & b_{3/2,j} & 0 & 0 \\ b_{3/2,j} & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & b_{n_x-\frac{1}{2},j} \\ 0 & 0 & b_{n_x-\frac{1}{2},j} & d_{n_x,j} \end{pmatrix} \tag{1.55}$$

e

$$T_{j+1/2} = \begin{pmatrix} c_{1,j+1/2} & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & c_{n_x,j+1/2} \end{pmatrix} \tag{1.56}$$

Per il problema numerico (1.51) i termini della diagonale principale di A sono moltiplicati per $\dfrac{1}{\Delta t}$ e i termini $T^l_{i,j}/\Delta t$, calcolati al tempo l, sono cogniti e perciò contribuiscono alla costruzione del termine noto $\mathbf{f}$. La matrice A è simmetrica e gode della proprietà di definita positività come è mostrato dal seguente teorema:

Teorema 1.3.1. *Se la funzione $a(x, y)$ è strettamente positivo sul dominio Ω allora la matrice A è definita positiva.*

Dimostrazione La matrice A è debolmente diagonalmente dominante, dal teorema di Gershgorin segue che i suoi autovalori sono maggiori o uguali a zero. Se si suppone un vettore $\mathbf{v}$ tale che $Av = 0$, posto con $v_{i,j}$ la componente di $\mathbf{v}$ di posizione $(j-1)n_x + i$, segue che $v_{i,j}$ può essere scritto nel modo seguente:

$$v_{i,j} = w_{i-1,j}v_{i-1,j} + w_{i+1,j}v_{i+1,j} + w_{i,j-1}v_{i,j-1} + w_{i,j+1}v_{i,j+1}, \tag{1.57}$$

dove

$$w_{i\pm1,j} \equiv \frac{1}{d_{i,j}}\frac{a_{i\pm1/2,j}}{h_x^2}, \quad w_{i,j\pm1} \equiv \frac{1}{d_{i,j}}\frac{a_{i,j\pm1/2}}{h_y^2}. \tag{1.58}$$

Osserviamo che i termini corrispondenti a nodi sulla frontiera valgono zero per le condizioni al bordo imposte al problema di diffusione (1.45). I pesi $w_{i\pm1,j}$ e $w_{i,j\pm1}$ sono positivi per le condizioni assunte sulla funzione a e la loro somma è uguale a 1. Se in prossimità della frontiera $\delta\Omega$ maggioriamo le componenti di v in (1.57) con

$\|v\|_\infty$ otteniamo un assurdo. L'assurdo deriva dall'avere supposto l'esistenza di un vettore $\mathbf{v} \neq 0$ per cui valga $A\mathbf{v} = 0$. Segue che la matrice A non può ammettere autovalori uguali a zero. ■

La risoluzione del problema di partenza (1.45) unito con le condizioni (1.46) si riduce dunque nella risoluzione di un sistema lineare sparso, simmetrico e definito positivo.

1.3.2 La Classe dei Metodi Iterativi

Un ben noto metodo di risoluzione per sistemi lineari è il metodo di eliminazione di Gauss. In generale questo richiede una complessità di spazio per la memorizzazione della matrice incompleta del sistema di ordine $O(n^2)$ locazioni di memoria ed una complessità di calcolo di ordine $O(n^3)$ operazioni floating point. Le matrici che però discendono da problemi pratici, come si è visto per la matrice A in (1.54), hanno un elevato grado di sparsità e questa caratteristica può essere solo in parte sfruttata dall'algoritmo di Gauss. Supposto che, ogni elemento $a(i,j) = 0$ in A con $|i-j| > m$, allora una versione a banda del algoritmo di Gauss può essere sfruttata. Memorizzando approssimativamente solo $2mn$ elementi all'interno della banda, cioè tutti gli elementi di A per cui $|i-j| \leq m$, la complessità di calcolo dell'algoritmo a banda è di circa $2m^2n$ operazioni floating point. Tuttavia questo tipo di algoritmo ancora non sfrutta la sparsità all'interno della banda distruggendo tale caratteristica ad ogni passo dell'algoritmo.

Al contrario la sparsità e altre proprietà delle matrici possono essere valorizzate maggiormente con l'uso del prodotto matrice per vettore. Se la matrice A ha solo pochi elementi diversi da zero per ogni sua riga, la complessità di spazio, con un opportuno formato di memorizzazione sparsa, è proprio con il numero degli elementi di A non nulli mentre la complessità di calcolo del prodotto ha ordine $O(n)$ operazioni floating point. Queste ragioni hanno portato alla ricerca di metodi risolutivi che utilizzano come operazione dominante il prodotto matrice per vettore al fine di trovare una buona approssimazione della soluzione dei sistemi lineare con un elevato indice di sparsità. Lo studio ha condotto alla classe dei metodi di tipo iterativo [90, 5, 51, 20, 34, 67, 97].

Dato un sistema lineare del tipo:

$$A\mathbf{x} = \mathbf{b}, \tag{1.59}$$

in generale gli algoritmi facenti parte della classe dei metodi di tipo iterativo costruiscono una successione approssimante $\{x_k\}_{k\in\mathbb{N}}$ della soluzione $\mathbf{x}$ di (1.59) che può essere rappresentato con la seguente successione definita per ricorrenza:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\big(\mathbf{b} - A\mathbf{x}_k\big) \tag{1.60}$$

con α_k parametro utilizzato per minimizzare o il residuo $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ nella norma euclidea o l'errore $\mathbf{e}_k = \mathbf{x} - \mathbf{x}_{k+1}$ nella A-norma se A è simmetrica e definita positiva, mediante appropriate tecniche di ortogonalizzazione. E' naturale scegliere, se non si ha nessuna ipotesi aggiuntiva sulla soluzione reale

$\mathbf{x}$, come primo termine della successione $\{x_k\}$ un multiplo di $\mathbf{b}$:

$$\mathbf{x}_0 \in < \mathbf{b} > . \tag{1.61}$$

Dopo aver calcolato il prodotto $A\mathbf{b}$ il prossimo elemento della successione si otterrà come combinazione lineare dei vettori $\mathbf{b}$ e $A\mathbf{b}$ ovvero:

$$x_2 \in < \mathbf{b}, A\mathbf{b} > . \tag{1.62}$$

Con lo stesso procedimento iterando, alla $k-$esima iterazione si conseguirà a:

$$x_k \in < \mathbf{b}, A\mathbf{b}, ..., A^{k-1}\mathbf{b} > . \tag{1.63}$$

Il sottospazio vettoriale di $\mathbb{R}^n$ in (1.63) è chiamato spazio di Krylov associato alla matrice A ed al termine noto $\mathbf{b}$. Molto spesso può accade che lo spazio di Krylov in (1.63) non contiene una buona approssimazione della soluzione $\mathbf{x}$ per un valore di k sufficientemente piccolo. Una maniera per risolvere questo problema consiste nel modificare il problema originale $A\mathbf{x} = \mathbf{b}$ trasformandolo in uno equivalente il cui spazio di Krylov associato contenga un buona approssimazione della soluzione $\mathbf{x}$ per valori di k sufficientemente piccoli. Un modo per per realizzare ciò si ottiene pensando al problema ottenuto moltiplicando a sinistra e a destra di (1.59) per l'inversa di una opportuna matrice P chiamata precondizionatore [12]. Il problema precondizionato diventa il seguente:

$$P^{-1}Ax = P^{-1}b \tag{1.64}$$

mentre la successione definita per ricorrenza che approssima la soluzione $\mathbf{x}$ si trasforma in:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha'_k P^{-1}\big(\mathbf{b} - A\mathbf{x}_k\big) \tag{1.65}$$

Ad ogni passo della nuova successione si aggiunge il costo del calcolo del vettore $\mathbf{z}_k = P^{-1}\mathbf{r}_k$ oppure del analogo problema mediante la risoluzione del sistema lineare seguente:

$$P\mathbf{z}_k = \mathbf{r}_k. \tag{1.66}$$

A tal fine la matrice P deve essere costruita in modo tale che il problema associato (1.66) sia più semplice di quello di partenza (1.59) di matrice incompleta A.

La ricerca di un opportuno precondizionatore è una obiettivo molto arduo sulla quale molti ricercatori hanno focalizzato il loro interesse in svariate classi di problemi, soprattutto quelli derivanti dalle differenze finite e dagli elementi finiti applicate alle equazioni differenziali alle derivate parziali.

Le matrici incomplete associate ai sistemi lineari derivanti da problemi di tipo pratico possono suddividersi in due categorie principali: quelle Hermtiane [1] e non Hermtiane se si considera matrici A a valori nel campo complesso più semplicemente simmetriche e non simmetriche se si vagliano matrici a valori reali. Le tecniche di costruzione di un opportuno precondizionatore, quando la matrice A è Hermtiana, ha molte soluzioni mentre nel caso opposto la costruzione di P rimane sostanzialmente una questione ancora aperta.

Se la matrice A è Hermtiana e il precondizionatore P è Hermtiano e defini-

[1] In algebra lineare una matrice hermitiana (o matrice autoaggiunta) è una matrice A a valori complessi che coincide con la propria trasposta coniugata A^H

to positivo allora invece di risolvere il sistema precondizonato a sinistra e a destra come in (1.64) si può implicitamente risolvere il seguente:

$$U^{-\top}AU^{-1}\mathbf{y} = U^{-\top}\mathbf{b} \quad \mathbf{x} = U^{-1}\mathbf{y} \tag{1.67}$$

con $P = U^{\top}U$ ed U il fattore di Cholesky di P. Ovviamente, come prima, non si deve realmente costruire la matrice $U^{-\top}AU^{-1}$, ma la successione approssimante la soluzione y nello spazio di Krylov associato a $U^{-\top}AU^{-1}$ e $U^{-\top}\mathbf{b}$. Se il precondizionatire P è indefinito allora il problema precondizionato non può essere trattato come in (1.67). In questo caso tecniche di precondizionamento per problemi non Hermitiani sono necessarie.

1.3.3 Solutori per Matrici Simmetriche

Per matrici reali e simmetriche o più in generale nel caso di matrici complesse e Hermtiane, diversi sono i metodi per determinare un' approssimazione della soluzione di (1.59) negli spazi di Krilov associati ad A e $\mathbf{b}$. Se un' approssimazione della soluzione di (1.59) si ricerca minimizzando il residuo $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ nella norma euclidea allora si può utilizzare l'algoritmo del Minimal Residual (MINRES) [83]. Mentre se la matrice A è anche definita positiva come quella in (1.52) si puo determinare una successione approssimante x nello spazio di Krylov associato ad A e b che minimizza la $A-$norma dell'errore $\|\mathbf{e}_k\| = (A^{-1}\mathbf{b}-\mathbf{x}_k, \mathbf{b}-A\mathbf{x}_k)^{\frac{1}{2}}$ e per tale scopo si può utilizzare l'algoritmo del Conjugate Gradient (CG) [61]. Per ognuno di questi algoritmi, l'ordine di grandezza della complessità di calcolo richiesta coincide con quel-

lo del prodotto matrice per vettore moltiplicato per il numero dii iterazioni, necessarie per generare la base dello spazio di Krylov da cui si otterra un approssimazione della soluzione. Mentre per quanto riguarda la complessità di spazio, pesa maggiormente la memorizzazione della matrice A ovviamente in un formato sparso come ad esempio il Compressed Row Storage (CRS), Compress Storage Column (CSC) oppure Coordinate List (COO) etc.
Per questi motivi i metodi iterativi vengono preferiti per risolvere sistemi lineari sparsi e simmetrici.
Inoltre questi metodi risolutivi hanno anche delle stime sulla velocità di convergente della successione approssimante generata in (1.60).
Il residuo alla $k-$esima iterazione del MINRES è $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$, scrivendo $\mathbf{x}_k$ come una combinazione lineare di vettori dello spazio di Krylov associato ad A e $\mathbf{b}$, allora otteniamo la seguente uguaglianza nella norma euclidea:

$$\|\mathbf{r}_k\| = \|P_k(A)\mathbf{b}\| \tag{1.68}$$

dove P_k è un polinomio di grado k con valore 1 nell'origine. Dalle proprietà di minimizzazione imposte sul residuo nel MINRES, per qualunque altro polinomio $p_k(A)$ con valore 1 nell'origine vale la seguente diseguaglianza:

$$\|r_k\| < \|p_k(A)\|\|b\| \tag{1.69}$$

Essendo A una matrice simmetrica, dal teorema spettrale si può scrivere $A = Q\Lambda Q^H$ dove $\Lambda = diag(\lambda_1, ..., \lambda_n)$ è la matrice diagonale degli autovalori di A, mentre Q è la matrice che ha sulle colonne gli autovettori di A. Dalla diseguaglianza in 1.70 otteniamo che:

$$\|r_k\| < \|Qp_k(A)Q^H\|\|b\| \leq \|p_k(\Lambda)\|\|b\| \tag{1.70}$$

Poichè questa diseguaglianza vale per ogni polinomio $p_k(\Lambda)$ che nell'origene vale 1, si ottiene :

$$\frac{\|r_k\|}{\|b\|} \leq \min_{p_k} \max_{i=1,\ldots,n} |p_k(\lambda_i)|. \tag{1.71}$$

Dalla diseguaglianza (1.72) segue che la norma del residuo relativo al passo k è completamenete determinata dagli autovalori della matrice A. Conoscendo una stima degli autovalori di A si può determinare la rapidità di convergenza della successione approssimate (1.60). Con un analogo ragionamento si dimostra quando A è definita positiva che per il CG vale un'analoga diseguaglianza per l'errore relativo nella A norma :

$$\frac{\|e_k\|_A}{\|A^{-1}b\|_A} \leq \min_{p_k} \max_{i=1,\ldots,n} |p_k(\lambda_i)|. \tag{1.72}$$

Allora per problemi Hermitiani come quello in 1.52 abbiamo dunque degli algoritmi per la quale si ha anche un idea di quanto buona sarà l'approssimazione basandosi sulle proprietà degli autovalori della matrice. Infine si aggiunge che questi solutori in aritmetica a precisione non sempre determinano una soluzione in aritmetica a precisione finita [50, 52, 37, 36].

Capitolo 2

Precondizionamento basato sull'Inversione della Decomposizione di Cholesky in un Modello Oceanico Globale

2.1 Introduzione

In questo capitolo, partendo dalle note tecniche di precondizionamento dei solutori iterativi [12] si analizza nel particolare il precondizionamento apportato ad un sistema lineare ottenuto dall'equazione ellittica di un modello di circolazione oceanica globale di nome NEMO [74]. In particolare si indaga sull'attuale metodologia di precondizionamento, rappresentata dal noto precondizionatore diagonale, per l'algoritmo del PCG, utilizzato dal modello

oceanico, per la risoluzione del sistema lineare in esame. Il nucleo ellittico del modello oceanico, è rappresentato da un problema di Laplace ed è discretizzato mediante differenze finite su una griglia finita di punti del dominio continuo del modello oceanico.

E'noto [96] che quando le risoluzioni delle griglie del modello oceanico aumentano, oppure, quando il dominio di simulazione include peculiari zone della Terra, come i poli terrestri, si ha una lenta convergenza del solutore iterativo utilizzato. Tuttavia, anche se la superficie e il volume degli oceani polari Artico e Antartico sono trascurabili rispetto al dominio globale del modello (la Terra) , le dinamiche oceaniche in questi luoghi hanno una grande influenza sulla circolazione marina globale. Ciò è dovuto ai processi di fusione e congelamento che coinvolgono queste aree e il conseguente scambio energia termica. Inoltre, queste aree costituiscono una grande riserva di acqua e sale, e quindi sono interessati anche da un elevato scambio di materia. Questi fatti portano a considerare seriamente i poli della Terra nel dominio di un modello oceanico globale.

Al fine di caratterizzare la lenta convergenza del solutore di OPA-NEMO per i motivi detti, si considera in principio il legame tra le prestazioni del precondizionatore diagonale, la dimensione del sistema lineare e i coefficienti del nucleo ellittico. In seguito, al fine di ottenere migliori prestazioni dal solutore si è proposto l'uso di una tecnica di precondizionamento, basata sull'inversione della decomposizione incompleta di Cholesky, che si rivelata essere un potente precondizionatore per il sistema lineare in disamina.

In generale il precondizionatore ottenuto dalla decomposizione incompleta di

Cholesky appartiene alla nota categoria dei precondizionatori di tipo Incomplete Lower Upper Decomposition (ILU). Questi precondizionatori furono introdotti negli anni 60, ma impiegati, in connessione agli spazi di Krylov solo alla fine degli anni 70 [76, 68], estesi e migliorati con il controllo del *fill in* durante la loro costruzione [12]. Il peggior svantaggio dei precondizionatori di tipo ILU, per grandi dimensioni, è la loro difficolta di parallelizzazione quando applicati per il precondizionamento del residuo ad ogni iterazione nel metodo iterativo utilizzato. Infatti, questi soffrono maggiormente della sequenzialità dell'algoritmo di backward e forward richiesti per la risoluzione di un sistema lineare, eseguiti ad ogni iterazione del solutore adoperato. Al contrario i precondizionatori inversi sparsi [12], già paralleli di natura, provvedendo ad una approssimazione esplicita P' di A^{-1}, alla quale si applica il matrice per vettore per il calcolo del precondizionamento del residuo $\widetilde{r}_k = P'r_k$ [7, 24, 57]. Per questa tipologio di precondizionatori si hanno 2 approcci principali: quelli basati sull'incompleta A-ortogonalizzazione [15], quelli basati sulla minimizzazione della norma di Frobenius [57].

Infine esiste un'ulteriore tecnica di precondizionamento che accoppia la metodologia ILU con quella degli inversi sparsi come in [65].

Il precondizioatore di Cholesky si è rivelato molto efficiente, dal punto di vista della rapidità di convergenza, per il sistema lineare associato al modello oceanico [43] quando questi include zone come i poli terrestri. Al fine di sfruttare anche l'efficienza computazionale dei precondizionatori inversi, si è costruito un nuovo precondizionatore $\widehat{P}'$, che è un'approssimazione sparsa dell'inversa della decomposizione di Cholesky, che conserva le proprietà ac-

celeranti di quest'ultimo sotto opportune assunzioni sulla matrice incompleta del sistema lineare in considerazione.

2.2 Il Nucleo Ellittico del Modello Oceanico

Lo stato dell'arte delle previsioni oceaniche è costituito da molti differenti modelli oceanici, se ne citano alcuni come Nemo [74], Hope [86], Sea [8], MOM [55] , POP [94] et al. [54]. Tutti i modelli oceanici menzionati si basano sulle seguenti equazione chiamate primitive [62, 79]:

$$\frac{\partial \mathbf{U}_h}{\partial t} = -\left[(\nabla \times \mathbf{U}) \times \mathbf{U} + \frac{1}{2}\nabla(\mathbf{U}^2)\right]_h - f\mathbf{k} \times \mathbf{U}_h - \frac{1}{\rho_0}\nabla_h(p_i + p_s) + \mathbf{D}^{\mathbf{U}} \quad (2.1)$$

$$\frac{\partial p_i}{\partial z} = -\rho g \quad (2.2)$$

$$\nabla \cdot \mathbf{U} = 0 \quad (2.3)$$

$$\frac{\partial T}{\partial t} = -\nabla \cdot (T\mathbf{U}) + D^T \quad (2.4)$$

$$\frac{\partial S}{\partial t} = -\nabla \cdot (S\mathbf{U}) + D^S \quad (2.5)$$

dove , U rappresenta il campo tridimensionale delle velocità, U_h è il campo delle velocità orizzontali, T, S, p_i sono rispettivamente la temperatura, salinità e pressione idrostatica. Infine, D^U, D^T e D^S rappresentano fenomeni di tipo diffusiovo o di vorticità e p_s rappresenta la pressione superficiale. Le differenze sostanziali tra questi modelli oceanici sono [54]:

- La rappresentazione dei fenomeni diffusivi e vorticità D^U, D^T e D^S in funzione di U, T e S e il calcolo del gradiente di pressione superficiale ∇p_s per la chiusura del modello alle primitive.

- La scelta delle coordinate verticali come la z coordinata , la ϱ coordinata e la σ coordinata e la scelta delle griglie orizzontali come le A-griglie, le B-griglie, le C-griglie e le E-griglie usate per la rappresentazione delle quantità fisiche del modello.

- L'uso di metodi numerici, come schemi temporali espliciti, semi-impliciti oppure impliciti, utilizzati per la discretizzazione delle equazioni evolutive (3.2), (2.4) e (2.5) nelle equazioni primitive.

Qui la nostra attenzione è rivolta al modello oceanico globale NEMO. Questo modello oceanico è principalmente composto da *engine* inseriti in un *environment*. Gli *engine* sono nuclei computazionali che forniscono le soluzioni numeriche per la simulazione delle variabili degli oceani, dei mari ghiacciati e della biochimica e della fisica a loro correlata. Queste engine sono chiamate rispettivamente OPA-NEMO, NEMO-LIM e NEMO-TOP [79]. L' *environment* è essenzialmente composto da strumenti di interfaccia del modello con le altre componenti del sistema climatico terrestre.
Nel particolare in questo capitolo si analizza il sistema lineare derivante dal nucleo ellittico del engine OPA-NEMO [74] e nella circostanza particolare, la tecnica di precondizionamento utilizzata per il solutore iterativo impiegato dal modello, rappresentato dal PCG.
Il nucleo ellittico in esame è rappresentato dal problema di Laplace seguente:

$$\begin{cases} \frac{\partial}{\partial x}\left[\alpha(x,y)\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial t}\right)\right] + \frac{\partial}{\partial y}\left[\beta(x,y)\frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial t}\right)\right] = f(x,y,t) \quad su\ \Omega \times [t_0,t_1] \\ \\ \frac{\partial \psi}{\partial t} = 0 \quad su\ \delta\Omega \times [t_0,t_1] \end{cases} \tag{2.6}$$

dove, la funzione incognita $\frac{\partial \psi}{\partial t}$ è chiamata Volume Transport Stream function. L'intervallo $[t_0, t_1]$ rappresenta l'intervallo di integrazione del modello oceanico e Ω è il suo dominio. Le funzioni α e β in (2.6) sono sempre positive e nel sistema di coordinate sferiche, essendo Ω una sfera, diventano $\alpha(\phi) = \frac{1}{H\cos\phi}$ e $\beta(\phi) = \frac{\cos\phi}{H}$ dove $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ è la longitudine. Il problema di Laplace (2.6) è fondamentale per il calcolo del gradiente di pressione superficiale ∇p_s presente nell'equazione di Nevier Stokes (3.2) e per il calcolo delle velocità barocliniche [74].

La discretizzazione di (2.6), per ogni passo temporale del modello $t_m = t_0 + m(t_1 - t_0)/N \quad m = 0, ...N$, è fatta per mezzo delle differenze finite nello spazio del secondo ordine su una griglia finita di punti $\Omega_{h,k}$ del dominio Ω [3]. Le differenze finite, per ogni passo temporale, restituiscono un sistema lineare $A\mathbf{x} = \mathbf{b}_m$ con $m = 0, ..., N$ dove con $\mathbf{x}$ si indica la funzione $\frac{\partial \psi}{\partial t}$ su $\Omega_{h,k}$. Ogni sistema $A\mathbf{x} = b_m$ è risolto, come già detto, mediante il PCG. Il nostro scopo è di caratterizzare numericamente la lenta convergenza del solver PCG quando si aumenta la risoluzione di $\Omega_{h,k}$ (vedi parte superiore della figura (2.1)), e quando il dominio del modello include alcune zone della Terra (vedi parte inferiore della figura (2.1)), quali i poli terrestri, per le simulazioni oceaniche [96].

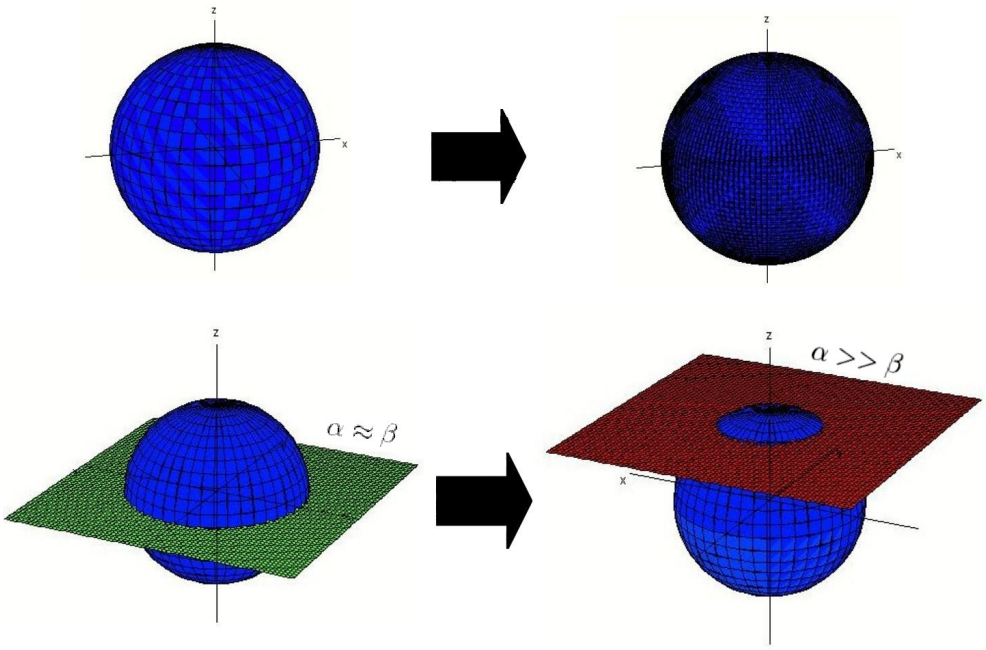

Figura 2.1: L' immagine mostra sfere con risoluzioni di griglia differenti sulla quale il rapporto dei coefficienti di Laplace del problema (2.6) varia come raffigurato

2.3 Preliminari sul Modello Numerico

Per ogni passo temporale, $t_m = t_0 + \frac{m}{N}(t_N - t_0)$ $m = 1, ..., N$, del modello oceanico, la discretizzaziona di (2.6) sulla griglia finita $\Omega_{h,k}$ di Ω, restituisce il seguente problema discreto:

$$\begin{cases} h^2\left(\alpha_{i+1,j}\frac{\partial\psi_{i+1,j}}{\partial t} - \left(\alpha_{i+1,j} + \alpha_{i-1,j}\right)\frac{\partial\psi_{i,j}}{\partial t} + \alpha_{i-1,j}\frac{\partial\psi_{i-1,j}}{\partial t}\right) + \\ +k^2\left(\beta_{i,j+1}\frac{\partial\psi_{i,j+1}}{\partial t} - \left(\beta_{i,j+1} + \beta_{i,j-1}\right)\frac{\partial\psi_{i,j}}{\partial t} + \beta_{i,j-1}\frac{\partial\psi_{i,j-1}}{\partial t}\right) \\ = (4hk)^2 f(i,j,t_m) \quad su\ \Omega_{k,h} \quad m = 1, ..., N \\ \frac{\partial\psi_{i,j}}{\partial t} = 0 \quad su\ \delta\Omega_{k,h} \quad m = 1, ..., N \end{cases} \tag{2.7}$$

dove, k e h sono i passi di dicretizzazione nelle direzioni $\vec{i}$ e $\vec{j}$. Per risolvere il modello numerico (2.7), su una griglia finita $\Omega_{k,h}$ dotata di $p \times q$ punti di Ω, al passo temporale t_m, si deve risolvere un sistema lineare:

$$A\mathbf{x} = b_m \tag{2.8}$$

dove la dimensione della matrice incompleta A è n, con $n = p \times q$ punti e $\mathbf{x}$ rappresenta la funzione $\frac{\partial \psi_{i,j}}{\partial t}$ su $\Omega_{k,h}$. La matrice A risulta essere sparsa, con un indice di sparsità $SP(A) < 1 - \frac{5}{n}$, simmetrica e definita positiva [74]. Gli elementi sulle sue diagonali sono i seguenti:

$$\begin{aligned} a_{m,m} &= h^2\big(\alpha_{i+1,j} + \alpha_{i-1,j}\big) + k^2\big(\beta_{i,j+1} + \beta_{i,j-1}\big) \\ a_{m,m-1} &= -h^2\alpha_{i-1,j}, \quad a_{m,m+1} = -h^2\alpha_{i+1,j} \\ a_{m,m-q} &= -k^2\beta_{i,j-1}, \quad a_{m,m+q} = -k^2\beta_{i,j+1} \end{aligned} \tag{2.9}$$

dove $m = 1, ..., n$ è legata al dominio discreto secondo la seguente relazione $m = (i-1)q + j$ con $i = 1, ..., p$ e $j = 1, ..., q$.

Per avere un' idea fino a quali dimensione si può giungere per A, si fa riferimento a una delle principali configurazioni del modello oceanico OPA-NEMO, denominata ORCA025 [53, 41], che impiega una griglia bidimensionale di 1440×1020 punti, quindi un ordine di grandezza per A di $O(10^6)$. Per risolvere il sistema (2.8), il modello oceanico NEMO usa due tipi di metodi iterativi: il PCG oppure il Successive Over Relaxation (SOR) [90]. Qui si indaga sul (PCG) poichè è un metodo rapido e semplice da utilizzare per un grande numero situazioni di tipo oceaniche [74] (fondo a topografia variabile, coste con geometria complessa, griglie con risoluzioni variabili, bordi ciclici o aperti). Inoltre, cosa essenziale è che il PCG non richiede la ricerca di un parametro di rilassamento opzionale come il SOR.

Il metodo del PCG determina, mediante una successione finita e definita per ricorrenza $\{\mathbf{x}_m\}$ $m = 0, ..., n$, un'approssimazione della soluzione $\mathbf{x}$ del sistema in (2.8) a meno di una tollernza ϵ prefissata sulla norma euclidea del residuo relativo.

$$res = \frac{\|A\mathbf{x}_k - b_m\|}{\|b_m\|} < \epsilon \tag{2.10}$$

dove x_k è il primo elemento della successione approssimante $\{\mathbf{x}_m\}$ che rispetta la tolleranza ϵ sul residuo. Il legame tra la rapidità di convergenza della successione approssimante $\{\mathbf{x}_m\}_m$ $m = 0, ..., n$ del Preconditioned Conjugate Gradient (PCG) e il numero di condizionamento $\mu(A)$ [1] è dato da seguente relazione:

$$\|\mathbf{x} - \mathbf{x}_m\|_A < 2\left(\frac{\sqrt{\mu(A)} - 1}{\sqrt{\mu(A)} + 1}\right)^{m-1} \|\mathbf{x} - \mathbf{x}_0\|_A \quad m = 0, .., n \tag{2.11}$$

Nell'equazione (2.11), si nota che se il numero di condizionamento $\mu(A)$ cresce allora la velocità della successione $\{\mathbf{x}_m\}_m$ $m = 0, ..., n$ decresce; mentre se $\mu(A)$ tende a 1, ovvero il numero di condizionamento associato alla matrice identica I, allora la velocità di convergenza aumenta. Poichè è noto [103] che la matrice A discendente da un problema ellittico mediante differenze finite ha un elevato numero di condizionamento, si necessita allora di tecniche di precondizionamento per incrementare la rapidità di convergenza del solutore

[1] Sia A una matrice invertibile in $R^{n\times n}$ e sia $||\ .\ ||$ una norma su uno spazio a dimensione finita. Si definisce numero di condizionamento di A la seguente quantità:

$$\mu(A) = ||A|| \cdot ||A^{-1}|| \quad .$$

iterativo utilizzato.

Al fine di studiare come la scelta del precondizionatore agisce sulla performance del solver richiamiamo qualche risultato di tipo teorico.

Teorema 2.3.1. *[49] Se F è una matrice invertibile appartenete a $\mathbb{R}^{n\times n}$ tale che $\|F\| < 1$ allora $I - F$, con I matrice unitaria, è invertibile e vale la seguente diseguaglianza:*

$$\|\big(I - F\big)^{-1}\| \leq \frac{1}{1 - \|F\|} \tag{2.12}$$

Teorema 2.3.2. *Sia $\{A_m\}_{\in\mathbb{N}}$ e A matrici invertibili appartenenti a $\mathbb{R}^{n\times n}$ tale che $A_m \to A$ per $m \to +\infty$ allora ${A_m}^{-1} \to A^{-1}$ per $m \to +\infty$.*

Dimostrazione Posto con $A_m = A + \Delta A_m$, per ipotesi, allora $\|\Delta A_m\|=\|A_m - A\| \to 0$ per $m \to +\infty$. Inoltre si può scrivere $A_m = A(I + A^{-1}\Delta A_m) = A\big(I - (-A^{-1}\Delta A_m)\big)$ dove si pone $F_m = -A^{-1}\Delta A_m$ e $\|F_m\| = \| - A^{-1}\Delta A_m\| \leq \|A^{-1}\|\|\Delta A_m\|$. Quindi $\|F_m\| \to 0$ per $m \to +\infty$, allora, esisterà un ν tale che $\forall m \geq \nu\ \|F_m\| < 1$. Dal teorema 2.3.1, segue che, $I - F_m$ per $m \geq \nu$ è invertibile e vale la diseguaglianza (2.12). Poichè $\|A_m^{-1}\| = \|\big(A(I-F_m)\big)^{-1}\| = \|(I-F_m)^{-1}A^{-1}\|$ allora si ottiene:

$$\|A_m^{-1}\| \leq \frac{\|A^{-1}\|}{1 - \|F_m\|} \quad \forall m \geq \nu \tag{2.13}$$

Poichè vale la seguente uguaglianza $A_m^{-1} = A^{-1} - A_m^{-1}\big(A_m - A\big)A^{-1}$ allora $\forall m \geq \nu$ segue che:

$$\|A_m^{-1} - A^{-1}\| \leq \frac{\|A^{-1}\|^2}{1 - \|F_m\|}\|\Delta A_m\| \quad \forall m \geq \nu \tag{2.14}$$

Per $m \to +\infty$ $\|F_m\|$ e $\|\Delta A_m\|$ tendono a zero da cui la tesi. ■

Teorema 2.3.3. *Sia* $\{A_m\}_{m\in\mathbb{N}}$ *una successione di matrici invertibili e* A *una matrice invertibile appartenente a* $\mathbb{R}^{n\times n}$ *con* n *finito tale che* $A_m \to A$ *per* $m \to +\infty$ *con* $\mu(A_m) \geq \mu(A)$. *Allora* $\mu(A_m) \to \mu(A)$ *per* $m \to +\infty$.

Dimostrazione Poichè $\mu(A_m) - \mu(A) = \|A_m\|\|A_m^{-1}\| - \|A\|\|A^{-1}\| \geq 0 \iff \|A + \Delta A_m\|\|A^{-1} + \Delta A_m^{-1}\| - \|A\|\|A^{-1}\| \geq 0$. Dalla diseguaglianza di Minkosky si ottiene che:

$$\begin{aligned} &\|A + \Delta A_m\|\|A^{-1} + \Delta A_m^{-1}\| - \|A\|\|A^{-1}\| \leq \\ &\leq \Big(\|A\| + \|\Delta A_m\|\Big)\Big(\|A^{-1}\| + \|\Delta A_m^{-1}\|\Big) - \|A\|\|A^{-1}\| = \end{aligned} \tag{2.15}$$

$$\|A\|\|\Delta A_m^{-1}\| + \|\Delta A_m\|\|A^{-1}\| + \|\Delta A_m\|\|\Delta A_m^{-1}\| \tag{2.16}$$

Poichè $A_m \to A$ allora anche $A_m^{-1} \to A^{-1}$ per $m \to +\infty$. Quindi la (2.16) va a 0 per $m \to +\infty$ e $\mu(A_m) \to \mu(A)$ per $m \to +\infty$. ■

Dal corollario (2.3.3) segue che bisogna scegliere un precondizionatore P tale che $P^{-1}A$ sia "vicino" alla matrice identica I, possibilmente, in una delle norme usuali per ottenere un numero di condizionamento $\mu(P^{-1}A)$ vicino l'unità. Il modello oceanico OPA-NEMO usa per precondizionare i sistemi lineari (2.8) il precondizionatore diagonale $P = diag(A)$.

2.4 Precondizionamento e Convergenza nel Modello Oceanico

I teoremi che vengono presentati in questo paragrafo forniscono delle stime, per il modello oceanico OPA NEMO, sulla distanza tra $P^{-1}A$, con P precondizionatore diagonale, ed I, matrice identica, nella norma di Frobenius legate alla dimensione n della griglia di risoluzione $\Omega_{h,k}$ ed ai coefficienti di Laplace nel caso di $p = q$ e di $k = h$, ovvero nel caso di un dominio quadrato con $n = p^2$.

Teorema 2.4.1. *[43] Se $P = diag(A)$ è il precondizionatore diagonale di A allora per la distanza $\|P^{-1}A - I\|_F$, valgono i seguenti limiti inferiore e superiore:*

$$\begin{aligned}&\sqrt{\tfrac{1}{2}\left(\tfrac{\alpha_{min}}{\alpha_{max}+\beta_{max}}\right)^2(n-1)+\tfrac{1}{2}\left(\tfrac{\beta_{min}}{\alpha_{max}+\beta_{max}}\right)^2(n-\sqrt{n})} \leq \\ &\leq \|P^{-1}A - I\|_F \leq \sqrt{\tfrac{1}{2}\left(\tfrac{\alpha_{max}}{\alpha_{min}+\beta_{min}}\right)^2(n-1)+\tfrac{1}{2}\left(\tfrac{\beta_{max}}{\alpha_{min}+\beta_{min}}\right)^2(n-\sqrt{n})}\end{aligned} \tag{2.17}$$

Dimostrazione Se precondizioniamo l'operatore lineare A mediante P otteniamo

la matrice $P^{-1}A$:

$$\begin{pmatrix}
1 & \frac{a_{1,2}}{a_{1,1}} & 0 & 0 & \frac{a_{1,q+1}}{a_{1,1}} & 0 & 0 & 0 & 0 \\
\frac{a_{2,1}}{a_{2,2}} & 1 & \frac{a_{2,3}}{a_{2,2}} & 0 & 0 & \frac{a_{2,q+2}}{a_{2,2}} & 0 & 0 & 0 \\
.. & .. & .. & .. & .. & .. & .. & .. & .. \\
\frac{a_{q+1,1}}{a_{q+1,q+1}} & .. & \frac{a_{q+1,q}}{a_{q+1,q+1}} & 1 & \frac{a_{q,q+2}}{a_{q,q}} & 0 & 0 & \frac{a_{q,2q}}{a_{q+1,q+1}} & 0 \\
.. & .. & .. & .. & .. & .. & .. & .. & .. \\
.. & .. & .. & .. & .. & .. & .. & .. & .. \\
0 & 0 & \frac{a_{n,n-q}}{a_{n,n}} & 0 & 0 & 0 & 0 & \frac{a_{n,n-1}}{a_{n,n}} & 1
\end{pmatrix}$$

e

$$\|P^{-1}A-I\|_F = \left\{ \left(\frac{a_{12}}{a_{11}}\right)^2 + \left(\frac{a_{1q+1}}{a_{11}}\right)^2 + \sum_{m=2}^{q} \left[\left(\frac{a_{mm-1}}{a_{mm}}\right)^2 + \left(\frac{a_{mm+1}}{a_{mm}}\right)^2 + \left(\frac{a_{mm+q}}{a_{mm}}\right)^2 \right] + \right.$$
$$+ \sum_{m=q+1}^{n-q} \left[\left(\frac{a_{mm-q}}{a_{mm}}\right)^2 + \left(\frac{a_{mm-1}}{a_{mm}}\right)^2 + \left(\frac{a_{mm+1}}{a_{mm}}\right)^2 + \left(\frac{a_{mm+q}}{a_{mm}}\right)^2 \right] +$$
$$\left. + \sum_{m=n-q+1}^{n-1} \left[\left(\frac{a_{mm-q}}{a_{mm}}\right)^2 + \left(\frac{a_{mm-1}}{a_{mm}}\right)^2 + \left(\frac{a_{mm+1}}{a_{mm}}\right)^2 \right] + \left[\left(\frac{a_{nn-q}}{a_{nn}}\right)^2 + \left(\frac{a_{nn-1}}{a_{nn}}\right)^2 \right] \right\}^{\frac{1}{2}} \tag{2.18}$$

Poichè le funzioni α e $\beta \in \mathcal{C}^1(\bar{\Omega})$ dove $\bar{\Omega}$ è un dominio chiuso e limitato di $\mathbb{R}^3$, dal teorema di Weistrass esisto α_{min}, α_{max}, β_{min} e β_{max} appartenenti a $\mathbb{R}$ tale che $\alpha_{min} \leq \alpha \leq \alpha_{max}$ e $\beta_{min} \leq \beta \leq \beta_{max}$ su $\mathcal{C}^1(\bar{\Omega})$, inoltre α e β hanno lo stesso segno. Utilizzando le espressioni in (3.10) si maggiora e minora $\|P^{-1}A-I\|_F$ come in (2.17). ■

Una prima analisi dei limiti inferiori e superiore in (2.17) mostrano che incrementando la dimensione del problema, la distanza tra $P^{-1}A$ e I cresce in

modo indefinito. Il numero di condizionamento $\mu(P^{-1}A)$ potrebbe dunque non essere più cosi vicino all'unità. La conseguenza è una perdita di velocità di convergenza del PCG. Un' analisi più approfondita di (2.17) rivela come la velocità di convergenza dipenda anche dal rapporto delle funzioni α e β del problema ellittico (2.6). Se si suppone che:

$$\alpha_{max}/\beta_{max} \approx 1 \quad \wedge \quad \alpha_{min}/\beta_{min} \approx 1 \tag{2.19}$$

e si pone $a = \frac{\alpha_{max}}{\alpha_{min}}$ e $b = \frac{\beta_{max}}{\beta_{min}}$, si ottiene la diseguaglianza seguente:

$$\sqrt{\frac{1}{8}\left(\frac{a^2+b^2}{a^2b^2}\right)n - \frac{1}{8b^2}\sqrt{n} - \frac{1}{8a^2}} \leq \|P^{-1}A - I\|_F \leq \sqrt{\frac{1}{8}(a^2+b^2)n - \frac{1}{8}b^2\sqrt{n} - \frac{1}{8}a^2}. \tag{2.20}$$

Se invece si ipotizza $\alpha_{max}/\beta_{min} << 1$ in (2.17) allora si consegue alla seguente:

$$\sqrt{\frac{1}{8b^2}(n-\sqrt{n})} \leq \|P^{-1}A - I\|_F \leq \sqrt{\frac{1}{2}b^2(n-\sqrt{n})} \tag{2.21}$$

L'ultimo caso si ottiene quando si pone $\beta_{max}/\alpha_{min} << 1$ in (2.17) dalla quale si ottiene la successiva diseguaglianza:

$$\sqrt{\frac{1}{8a^2}(n-1)} \leq \|P^{-1}A - I\|_F \leq \sqrt{\frac{1}{2}a^2(n-1)} \tag{2.22}$$

Le stime in (2.20), (2.21) e (2.22) mostrano che si hanno differenti velocità di convergenza del solutore anche quando il rapporto tra le funzioni α e β varia fissata la dimensione n del problema. La rapidità di convergenza è maggiore quando $\alpha_{max} \approx \beta_{max} \wedge \alpha_{min} \approx \beta_{max}$ o $\alpha_{max}/\beta_{min} << 1$ poichè $\|P^{-1}A - I\|_F^2$ è un $\mathcal{O}(n - \sqrt{n})$ mentre la velocità di convergenza decresce quando $\beta_{max}/\alpha_{min} << 1$ in quanto $\|P^{-1}A - I\|_F^2$ è un $\mathcal{O}(n)$. Questi risultati teorici verranno supportati da esperimenti numerici condotti nel paragrafo (2.6).

La perdita della rapidità di convergenza del solutore influisce anche sull'accuratezza della soluzione, dovuta alla propagazione dell'errore di round-off nei calcoli, quando il PCG è eseguito su un sistema a precisione finita. Infatti seguendo le metodologie dell'analisi della propagazione dell'errore come in [30, 29, 31] per lo schema dell'algoritmo in (1) tratto da [34], possiamo supporre che alla m-esima iterazione, il vettore $\mathbf{x}_m$ sia affetto da un errore $\epsilon_{\mathbf{x}}(m)$, il termine η_m in schema (1) sia affetto da un errore relativo δ_m con $|\delta_m| \leq u$ ovvero $\tilde{\eta}_k = \eta_m(1 + \delta_m)$ con u massima accuratezza relativa e $\mathbf{d}(m)$ sia affetto dall' errore $\epsilon_{\mathbf{d}}(m)$. Al passo $m + 1$ si ottiene che:

$$\tilde{\mathbf{x}}_{m+1} = \tilde{\mathbf{x}}_m + \tilde{\eta}_m \tilde{d}_m = \mathbf{x}_k + \epsilon_{\mathbf{x}}(m) + \eta_m(1 + \delta_m)(\mathbf{d}_m + \epsilon_{\mathbf{d}}(m)) = \mathbf{x}_{m+1} + \big[\epsilon_{\mathbf{x}}(m) + \\ + \eta_m\big(\delta_m \mathbf{d}_m + (1 + \delta_m)\epsilon_{\mathbf{d}}(m)\big)\big] \tag{2.23}$$

L'errore sulla successione $\{\mathbf{x}_m\}$ alla $(m + 1)$-esima iterazione è data da

$$\epsilon_{\mathbf{x}}(m + 1) = \big[\epsilon_{\mathbf{x}}(m) + \eta_m\big(\delta_m \mathbf{d}_m + (1 + \delta_m)\epsilon_{\mathbf{d}}(m)\big)\big] \tag{2.24}$$

Schema 1 Algoritmo del Gradiente Coniugato Precondizioanto

$iter \leftarrow 0$

$x_0 \leftarrow b$

$r_0 \leftarrow b - Ax_0$

$\widehat{r}_0 \leftarrow P^{-1} r_0$

$d_0 \leftarrow z_0$

repeat

$iter \leftarrow iter + 1$

$$\eta_k \leftarrow \frac{\widehat{r}_k^{\top} r_k}{d_k^{\top} A d_k}$$

$x_{k+1} \leftarrow x_k + \alpha_k d_k$

$r_{k+1} \leftarrow r_k - \alpha_k A d_k$

$\widehat{r}_k \leftarrow P^{-1} r_k$

$$\omega_k \leftarrow \frac{\widehat{r}_{k+1}^{\top} r_{k+1}}{\widehat{r}_k^{\top} r_k}$$

$d_{k+1} \leftarrow \widehat{r}_{k+1} + \beta_k d_k$

until $\left(\dfrac{\|r_{k+1}\|}{\|b\|} < \epsilon \ .or.\ iter \geq n \right)$.

che è somma dell' errore $\epsilon_{\mathbf{x}}(m)$ con l'addendo $\eta_m \big(\delta_m \mathbf{d}_m + (1+\delta_m)\epsilon_{\mathbf{d}}(m)\big)$ dove $\eta_m \epsilon_{\mathbf{d}}(m)$ è in norma la quantità con il più grande ordine di grandezza per m grande. Conseguentemente quando la dimensione n del sistema lineare è molto grande l'errore $\epsilon_{\mathbf{x}}(m)$ può crescere in modo tale da rendere $\mathbf{x}(m)$ totalmente inaccettabile.

2.5 Un Precondizionatore Inverso Sparso nel Modello Oceanico

A seguito dei risultati pervenuti in (2.17) (2.20), (2.21) e (2.22), al fine di incrementare la velocità di convergenza del PCG nel modello OPA-NEMO quando la risoluzione del dominio $\Omega_{k,h}$ aumenta e soprattutto quando α è maggiore di β, essendo A una matrice simmetrica e definita positiva, si sostituisce il precondizionatore diagonale P con un precondizionatore basato sull'inversione della fattorizzazione incompleta di Cholesky di A [15]. La decomposizione incompleta di Chlesky [68, 75, 100, 73] è un potente precondizionatore per accelerare la rapidità di convergenza del problema dato [43] come anche riportato nei test fatti nel paragrafo dedicato agli esperimenti numerici (2.6). La fattorizzazione di Cholesky è data da:

$$\widehat{P} = U^t \cdot U. \tag{2.25}$$

dove U è una matrice triangolare superiore con la stessa struttura di sparsità della parte triangolare superiore di A. Poichè A è una matrice sparsa ($SP(A) \leq 1 - \frac{5}{n}$) allora l'algoritmo della decomposizione di Cholesky per il calcolo di U ha una complessità di tempo e di spazio in termini di costo computazionale dell' ordine di $\mathcal{O}(n)$. Inoltre per l'implementazione di $\widehat{P}$ nel PCG, essendo $z_m = \widehat{P}^{-1} r_m \iff \widehat{P} z_m = r_m$, per la risoluzione del sistema lineare $\widehat{P} z_m = r_m \Leftrightarrow U^t U z = r_m$ nel PCG, si necessita di una forward substitution e una backward substitution. Le complessità di tempo di questi risolutori triangolari nel caso specifico sono ancora un $O(n)$ data la sparsità

di U. Lo svantaggio di $\widehat{P}$, sta nell'utilizzo per grandi dimensioni di A, a causa della loro difficoltà di parallelizzazione, causata dalla sequenzialità dell'algoritmo di backward e forward richiesti per la risoluzione del sistema $\widehat{P}z_k = r_k$ ad una k-esima iterazione del solutore. Un precondizionatore di tipo inverso P' supera questa difficoltà utilizzando, al posto della risoluzione di sistemi lineari, il prodotto matrice per vettore per il precondizionamento del residuo $z_k = P'r_k$. Infatti Il prodotto matrice per vettore è un operazione agevole da parallelizzare e molto efficiente sopratutto sulle nuove architetture di calcolo basate sul Graphic Processing Unit (GPU) computing [10, 46, 18].
Per la costruzione del precondizionatore inverso $\widehat{P'}$, basato sul Cholesky, daremo infine dei risultati teorici che possono darci informazioni su come ridurre il *fill in* della matrice U^{-1} per preservare la sparsità del problema. Introduciamo qui alcuni teoremi che serviranno a tale proposito.

Proposizione 2.5.1. *La matrice incompleta A del sistema lineare (2.8) è diagonalmente dominante:*

$$|a_{m,m}| \geq \sum_{s \neq m} |a_{m,s}| \qquad m = 1, ..., n. \tag{2.26}$$

Dimostrazione Essendo α e β in (2.6) funzioni sempre positive su Ω segue che: $\forall m \in \{1, ..., n\}$ posto $i \in \{1, ..., p\}$ e $j \in \{1, ..., q\}$ tale che $m = (i-1)q + j$ allora valgono le seguenti uguaglianze

$$\begin{aligned} |a_{m,m}| &= |h^2\big(\alpha_{i+1,j} + \alpha_{i-1,j}\big) + k^2\big(\beta_{i,j+1} + \beta_{i,j-1}\big)| = \\ &|h^2\alpha_{i+1,j}| + |h^2\alpha_{i-1,j}| + |k^2\beta_{i,j+1}| + |k^2\beta_{i,j-1}| = \\ &= |a_{m,m+1}| + |a_{m,m-1}| + |a_{m,m+q}| + |a_{m,m-q}|. \end{aligned} \tag{2.27}$$

cioe la tesi. ■

Dalla valutazione della distanza $\|P^{-1}A - I\|_F$ fatta in (2.22) con P precondizionatore diagonale e dagli esperimenti numerici condotti nel paragrafo (2.6), i cui risultati sono mostrati in tabella (2.1), si può osservare che la rapidità di convergenza decresce in modo significativo quando la funzione α è molto maggiore della funzione β. In tal caso però la matrice A può essere approssimata con una matrice tridiagonale. Sotto questa assunzione per A mostriamo il seguente risultato

Proposizione 2.5.2. *Se A è una matrice tridiagonale, simmetrica e definita positiva allora il fattore incompleto di Cholesky U è composto dalle seguente diagonali principale e prima diagonale superiore:*

$$\begin{aligned} u_{m,m} &= \sqrt{\frac{\det A_m}{\det A_{m-1}}} \quad m = 2, ..., n, \\ u_{m,m+1} &= \sqrt{\frac{detA_{m-1}}{\det A_m}}\ a_{m,m+1}\ m = 2, ..., n-1 \end{aligned} \tag{2.28}$$

dove $u_{1,1} = \sqrt{a_{1,1}}$, $u_{1,2} = a_{1,2}/\sqrt{a_{1,1}}$ e $\det A_m$, $m = 1, ...n$ sono i determinanti dei minori principali di A.

Dimostrazione La dimostrazione segue per induzione mediante l'algoritmo del calcolo del fattore triangolare incompleto superiore di Cholesky [68, 73, 75]. ■

Proposizione 2.5.3. *Il fattore incompleto di Cholesky U della matrice A è a diagonale dominante:*

Dimostrazione Per dimostrare la tesi è sufficiente provare che:

$$\left|\frac{u_{m,m+1}}{u_{m,m}}\right| < 1 \quad m = 1, ..., n. \tag{2.29}$$

Per $m = 1$ la verifica è banale. Per $m = 2, ..., n$, valendosi delle uguaglianze in (2.28), determinate nel teorema 2.5.2, otteniamo:

$$\left|\frac{u_{m,m+1}}{u_{m,n}}\right| = \frac{detA_{m-1}}{\det A_m}\ a_{m,m+1} = \left[\det A_{m-1}/\left[(-1)^{2m-1}a_{m,m-1}\det A_m^{m,m-1} + (-1)^{2m}a_{m,m}\det A_m^{m,m}\right]\right]\ a_{m,m+1}. \tag{2.30}$$

dove $(-1)^{2m-1}\det A_m^{m,m-1}$ è il complemento algebrico di $a_{m,m-1}$ in A_m mentre $(-1)^{2m}\det A_m^{m,m}$ è il complemento algebrico di $a_{m,m}$ e coincide con $\det A_{m-1}$. E' banale verificare che nella matrice tridiagonale A, la quantità $(-1)^{2m-1}a_{m,m-1}detA_m^{m,m-1}$ è sempre positiva per ogni $m = 2, ...n$. Inoltre essendo $a_{m,m} \geq a_{m,m-1}$, poichè A è a diagonale dominante, segue che $\left|\frac{u_{m,m+1}}{u_{m,n}}\right| < 1$ cioè la tesi. ■

Proposizione 2.5.4. *L 'inversa del fattore incompleto di Cholesky U^{-1} della matrice A è costituita per ogni $m = 1, ..., n$ da successioni $u^{-1}_{m-i,m}$, $i = 0, , , , m-1$ strettamente decrescenti.*

Dimostrazione Iniziamo con l'osservare che l'inversa U^{-1} del fattore incompleto di Cholesky U è formato dai seguenti elementi:

$$\left|u^{-1}_{m,m+s}\right| = \begin{cases} \left|\frac{1}{u_{m,m}}\right| & if\ s = 0 \\ \left|\frac{1}{u_{m,m}}\right| \prod_{r=1}^{s}\left|\frac{u_{m+r-1,m+r}}{u_{m+r,m+r}}\right| & if\ 0 < s \leq n-m \end{cases} \tag{2.31}$$

mentre la parte triangolare inferiore è costituita da tutti elementi nulli come mostrato in figura. Segue che per m=1,...,n la successione $u^{-1}_{m-i,m}$, $i = 0, , , , m-1$ è definita nel seguente modo:

$$|u^{-1}_{m-i,m}| = \begin{cases} \left|\dfrac{1}{u_{m,m}}\right| & if\ i = 0 \\ \left|\dfrac{1}{u_{m,m}}\right| \displaystyle\prod_{r=1}^{i} \left|\dfrac{u_{m-r,m-r+1}}{u_{m-r,m-r}}\right| & if\ 0 < i \leq m-1 \end{cases} \tag{2.32}$$

Dalla proposizione 2.5.3, essendo $\left|\frac{u_{m,m+1}}{u_{m,m}}\right| < 1 \quad m = 1, ..., n.$, segue la tesi. ■

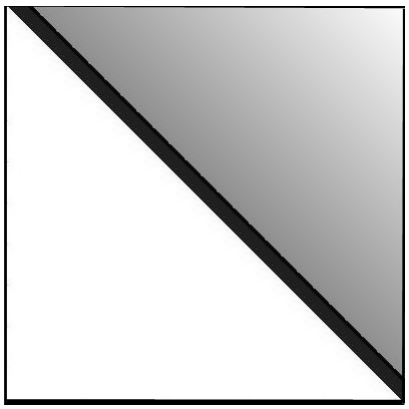

Figura 2.2: Con le tonalità di grigio, l' immagine rappresenta l'ordine di grandezza degli elementi della matrice U^{-1}, evidenziando con le tonalità più scure quelli in valore assoluto con più alto ordine di grandezza.

In generale se A è una matrice simmetrica e definita positiva, allora la matrice U^{-1} può essere ottenuta per mezzo di una base di vettori $A-$ortonormale $\{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_n\}$ di $\mathbb{R}^n$. Se indichiamo con $Z = [\mathbf{z}_1, ...\mathbf{z}_n]$, la matrice la cui $i-$esima colonna coincide con il vettore $\mathbf{z}_i$, allora si ricava che:

$$Z^\top A Z = I \tag{2.33}$$

da cui segue che:

$$A = Z^{-\top} Z^{-1} \quad e \quad A^{-1} = ZZ^{\top}. \tag{2.34}$$

Per mezzo della (2.33) si ottiene cosi una fattorizzazione di A^{-1}. La matrice Z può essere ottenuta per opera dell' algoritmo della fattorizzazione inversa [15, 14] in schema (2) basato sulla A ortogonalizzazione derivante dalla procedura di Gram Schmidt, riportato in schema (3).

Schema 2 Algoritmo della Fattorizzazione Inversa

$\omega_i^0 = e_i \quad 1 \leq i \leq n$

for $i = 1 \rightarrow n$ **do**

for $j = i \rightarrow n$ **do**

$p_j^{(i-1)} = a_i^T \omega_j^{(i-1)}$

end for

if $i < n$ **then**

for $j = i + 1 \rightarrow n$ **do**

$$\omega_j^{(i)} = \omega_j^{(i-1)} - \frac{p_j^{(i-1)}}{p_i^{(i-1)}} \omega_i^{(i-1)}$$

end for

end if

end for

$$\Omega = \left[\omega_1, ..., \omega_n\right] \quad D = \begin{pmatrix} p_1 & \cdots & \cdots & \cdots \\ \cdots & \ddots & \cdots & \cdots \\ \cdots & \cdots & \ddots & \cdots \\ \cdots & \cdots & \cdots & p_n \end{pmatrix} \Longrightarrow Z = \Omega D^{-\frac{1}{2}}$$

L'algoritmo di $A-$ortogonalizzazione può essere applicato a qualsiasi sistema di vettori linearmente indipendente massimale $\{\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_n\}$ di $\mathbb{R}^n$. Nel caso

particolare che il sistema di generatori minimale di partenza coincida con la base naturale $\{\mathbf{e}_1, ..., \mathbf{e}_n\}$ di $\mathbb{R}^n$ allora la matrice risultante Z, per mezzo dell'algoritmo di fattorizzazione inversa, è una matrice triangolare superiore. Discende da ciò che Z^{-1} coincide con il fattore di Cholesky di A e dunque la matrice Z con l'inversa del fattore di Cholsky U^{-1}.

Schema 3 Procedura di $A-$ortonormalizzazione

$z_1 = e_1$

for $k = 2 \rightarrow n$ **do**

$\widetilde{z}_k = e_k$

for $i = 1 \rightarrow k-1$ **do**

$\widetilde{z}_k \leftarrow \widetilde{z}_k - (\widetilde{z}_k, Az_i)z_i$

end for

$z_k = \dfrac{\widetilde{z}_k}{\|\widetilde{z}_k\|_A}$

end for

L' algoritmo di fattorizzazione inversa fu proposto per la prima volta in [1] per la risoluzione di sistemi lineari $A\mathbf{x} = \mathbf{b}$, con A matrice simmetrica e definita positiva attraverso la formula:

$$\mathbf{x} = A^{-1}\mathbf{b} = ZZ^T\mathbf{b} \tag{2.35}$$

Per ulteriori riferimenti si vedano anche i testi [42, 63]. In generale per una matrice densa questo metodo richiede una complessità computazionale di circa il doppio di quella della fattorizzazione di Cholesky. Per una matrice sparsa il costo può essere sostanzialmente ridotto, ma il metodo è ancora

impraticabile, per elevate dimensioni n, perché la matrice risultante Z è tendenzialmente una matrice triangolare superiore densa. Inseguito l'algoritmo fu utilizzato per calcolare un' approssimazione sparsa di Z per costruire un precondizionatore per A, questa idea fu proposta per prima in [11]. Altri riferimenti bibliografici dove si utilizzano questa metodologia di calcolo per la costruzione di precondizionatori sono [13, 48, 15]. Le motivazioni per una costruzione sparsa di Z, di massima, sono basate su risultati teorici ed esperimenti al computer che mostrano come molti degli elementi non nulli di Z di una matrice sparsa simmetrica e definita positiva A sono piccoli in valore assoluto [6, 33, 78]. Diversi autori hanno sfruttato questa caratteristica per costruire precondizionatori espliciti sulla base dei precondizionatori inversi sparsi [6, 70, 40].

Per i sistemi lineari in (2.8) del modello oceanico, riprendendo la proposizione (2.5.4), la quale garantisce che le colonne di U^{-1}, a partire dall'elemento diagonale, costituiscono delle successioni strettamente decrescenti, allora la sparsità e la complessità di calcolo sono preservate riducendo la quantità di *fill-in* al di sopra della diagonale principale di Z durante la sua computazione.

Per non incrementare la complessità di spazio totale del problema si può costruire Z con la stessa struttura di sparsità della parte triangolare superiore di A che supporremo d'ora in avanti "tridiagonale". Si ricorda che questa supposizione è dovuta al fatto che,a seguito dei risultati teorici in (2.22) e degli esperimenti numerici condotti in (2.6), è proprio quando la funzione α è molto maggiore di β che la rapidità di convergenza del PCG decresce

enormemente, ma per le eguaglianze in (3.10), in tal caso, si può approssimare A proprio con una matrice tridiagonale.

L'algoritmo utilizzato per la costruzione di Z, il cui schema è rappresentato in (4), è scritto in linguaggio C ed è basato anche esso sulla procedura di Gram Schmidt in (3) e la fattorizzazione inversa in schema (4), dove la matrice A e Z vengono memorizzate utilizzando rispettivamente il formato di registrazione CSR e il CSC in modo tale da preservare la complessita totale del problema ad un'ordine di grandezza di $O(n)$.

Durante la costruzione dei vettori $\mathbf{z}_i \quad i = 2, ..., n$ di Z, volendo costruire una matrice risultante con la solo diagonale principale e la prima diagonale superiore, allora per ogni $\mathbf{z}_i$ si lavora solo sulle sue componenti $z_i[i-i]$ e $z_i[i]$ per $i = 2, ..., n$.

Al primo passo dell'algoritmo in (4), nello specifico, si $A-$normalizza il vettore $\mathbf{e}_1$ mediante il calcolo della $A-$norma $\|\mathbf{e}_1\|_A = \sqrt{(\mathbf{e}_1, A\mathbf{e}_1)}$, dove però si calcola solo la prima componente del vettore $\mathbf{y} = A\mathbf{e}_1$ dovendo essere moltiplicato scalarmente per il vettore $\mathbf{e}_1 = (1, 0, ..., 0)$. L'algoritmo segue poi con la fase di $A-$ortogonalizzazione e di $A-$normalizzazione dei vettori $\mathbf{e}_i \quad i = 2, ..., n$ dove $\mathbf{e}_i$ viene $A-$ortogonalizzato solo con il vettore $\mathbf{z}_{i-1}$ essendo per costruzione già $A-$coniugato con tutti i vettori $\mathbf{z}_j$ con $j < i-1$. La $A-$ortogonalizzazione tra $\mathbf{e}_i$ e $\mathbf{z}_{i-1}$ si ottiene attraverso la seguente operazione vettoriale:

$$\mathbf{z_i} = \mathbf{e}_i - (\mathbf{e}_i, A\mathbf{z}_{i-1})\mathbf{z}_{i-1} \tag{2.36}$$

Per il calcolo di $\mathbf{y} = A\mathbf{z}_{i-1}$ è occorrente calcolare solo la $i-$esima componente di $\mathbf{y}$, dovendo poi essere moltiplicato per il vettore della base naturale $\mathbf{e}_i$.

Infine, comunque scelto un vettore $\mathbf{z}_i \quad i = 2, ..., n$, la fase di normalizzazione si consegue per mezzo della seguente formula:

$$\widetilde{\mathbf{z}_i} = \frac{\mathbf{z}_i}{\sqrt{(\mathbf{z}_i, Az_i)}} \tag{2.37}$$

calcolando solo la $i-1$ e la $i-$esima componente di $\mathbf{y} = A\mathbf{z}_i$ essendo successivamente moltiplicato scalarmente per il vettore $\mathbf{z}_i$. La complessità di calcolo dell'algoritmo in schema (4) nello schema per la costruzione dell'inversa del fattore di Cholesky ha dunque un ordine di grandezza $O(n)$.

Schema 4 Nucleo dell' Algoritmo della Fattorizzazione Inversa

```
myArr = (double *)calloc( n, sizeof(double) );
// A- normalizzazione di $z_1$
myArr[0]=z[0];
h=irow_z[start_z[0]];
y[h]=0;
for(k=start_a[h];k<start_a[h+1];k++)
    y[h]=y[h]+a[k]*myArr[jcol_a[k]-1];
alpha=myArr[0]*y[0];
alpha=sqrt(alpha);
alpha=1/alpha;
z[0]=alpha*z[0];
//Fase di ortogononalizzazione $z_1,z_2,...z_n$
for(i=1;i<n;i++)
    free(myArr);
    myArr = (double *)calloc( n, sizeof(double) );
    for(k=start_z[i-1]; k<start_z[i];k++)
        myArr[irow_z[k]]=z[k];
    h=irow[start_z[i]+1];
    y[h]=0;
    for(k=start_a[h];k<start_a[h+1];k++)
        y[h]=y[h]+a[k]*myArr[jcol_a[k]-1];
    alpha=0;
    alpha=alpha+z[start_z[i]+1]*y[irow_z[start_z[i]+1]];
    alpha=-alpha;
    z[start_z[i]]= z[start_z[i]]+alpha*z[start_z[i]-1];
    //Fase di normalizzazione del vettore $z_i$
    free(myArr);
    myArr = (double *)calloc( n, sizeof(double) );
    for(k=start_z[i]; k<start_z[i+1];k++)
        myArr[irow_z[k]]=z[k];
    for(t=start_z[i]; t<=start_z[i]+1;t++)
       {h=irow[t];
        y[h]=0;
            for(k=start_a[h];k<start_a[h+1];k++)
                y[h]=y[h]+a[k]*myArr[jcol_a[k]-1];}
    alpha=0;
    for(t=start_z[i]; t<=start_z[i]+1;t++)
    alpha=alpha+z[t]*y[irow_z[t]];
    alpha=sqrt(alpha);
    alpha=1/alpha;
    for(k=start_z[i]; k<start_z[i+1];k++)
        z[k]=alpha*z[k];
```

2.6 Esperimenti Numerici

In questa sezione si riportano i risultati dell'esecuzione del solutore PCG con il precondizionatore diagonale P, con il precondizionatore di Cholesky e con il precondizionatore inverso $\widehat{P}' = ZZ^\top$ definito nel paragrafo precedente (2.5) costruito per mezzo dell'algoritmo in schema (4) su di un `Intel Core 2 Duo Processor T5450 (2M Cache, 1.66 GHz, 667 MHz FSB)`.

Per validare le osservazioni teoriche fatte in (2.17) (2.20), (2.21) e (2.22) si inizia con l' esecuzione del PCG con precondizionatore diagonale P per una matrice A' con le stesse proprietà della matrice A dei sistemi in (2.8), aumentando di volta in volta la dimensione n di un fattore 10 nell'intervallo cha va da $n = 100$ a $n = 100000$. Si investiga soprattutto sulla rapidità di convergenza e sul tempo che il PCG impiega per giungere a tolleranza quando il rapporto tra le funzioni α e β varia. Negli esperimenti numerici qui di seguito si assegna una tolleranza sul residuo res:

$$res = \frac{\|Ax_m - b\|}{\|b\|} < tol = 10^{-6} \quad m = 0, ..., n,$$

dove $\|.\|$ è la norma Euclidea.

Il software che implementa il PCG è scritto in linguaggio C e utilizza il formato di memorizzazione sparsa CSR per la matrice A'. Nella tabella (2.1) viene riportato il numero di iterazioni , $iter$, necessario per ottenere una tolleranza posta sul residuo tol, al variare della dimensione n e del rapporto tra le funzioni α e β nel caso del precondizionatore diagonale. Si riportano nella

tabella (2.1) anche i tempi in secondi per la costruzione del precondizionatore P, indicato con $time_P$ e per la risoluzione del sistema precondizionato indicato con $time_{\alpha<<\beta}$, $time_{\alpha\approx\beta}$ e $time_{\alpha>>\beta}$ rispettivamente quando $\alpha << \beta$, $\alpha \approx \beta$ e $\alpha >> \beta$.

n	$iter_{\alpha<<\beta}$	$iter_{\alpha\approx\beta}$	$iter_{\alpha>>\beta}$	$time_P$	$time_{\alpha<<\beta}$	$time_{\alpha\approx\beta}$	$time_{\alpha>>\beta}$
100	40	38	100	$0,2 \times 10^{-3}$	$0,1 \times 10^{-1}$	$0,1 \times 10^{-1}$	$0,3 \times 10^{-1}$
1000	100	100	1000	$0,2 \times 10^{-2}$	$0,2 \times 10^{0}$	$0,2 \times 10^{0}$	$0,2 \times 10^{1}$
10000	734	616	5671	0.2×10^{-1}	$0,1 \times 10^{2}$	$0,8 \times 10^{1}$	$0,78 \times 10^{2}$
100000	739	568	6214	0.2×10^{0}	$0,11 \times 10^{3}$	$0,86 \times 10^{2}$	0.70×10^{3}

Tabella 2.1: Nel prospetto la variabile *iter* è il numero di iterazioni che il solutore impiega nel tempo *time*, per raggiungere la tolleranza di 10^{-6} posta sul residuo *res*, con precondizionatore diagonale P, infine n è la dimensione della matrice incompleta A'.

Gli esperimenti numerici riportati in tabella (2.1) confermano le osservazioni teoriche fatte in (2.20), (2.21) e (2.22). La velocità di convergenza del PCG decresce incrementando la dimensione n di A'. Inoltre, le esperienze a calcolatore confermano che la velocità di convergenza del solutore dipende anche dal rapporto delle funzioni α e β, infatti fissata la dimensione n e la tolleranza $tol = 10^{-6}$, essa è più grande quando l'ordine di grandezza della funzione α è minore di quella della funzione β $\left(\alpha_{max} << \beta_{min}\right)$ e quando sono dello stesso ordine $\left(\alpha \approx \beta\right)$ mentre decresce significativamente quando l'ordine di grandezza della funzione α è molto maggiore di quello della funzione β $\left(\alpha_{min} >> \beta_{max}\right)$ come mostrato in figura (2.3) nel caso di $n = 100000$.

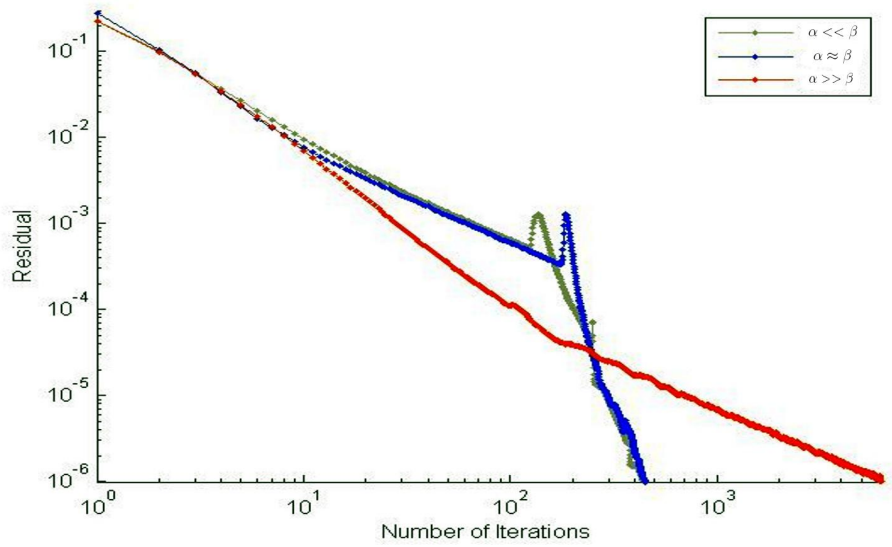

Figura 2.3: L'immagine mostra il comportamento del residuo relativo *res* al crescere del numero di iterazioni del PCG con precondizionatore diagonale quando $\alpha << \beta$ (verde) and $\alpha \approx \beta$ (blu) e $\alpha >> \beta$ (rosso) per $n = 100000$

n	$iter_{\alpha<<\beta}$	$iter_{\alpha\approx\beta}$	$iter_{\alpha>>\beta}$	$time_P$	$time_{\alpha<<\beta}$	$time_{\alpha\approx\beta}$	$time_{\alpha>>\beta}$
100	2	15	3	$0,2 \times 10^{-2}$	$0,2 \times 10^{-2}$	$0,1 \times 10^{-1}$	$0,2 \times 10^{-2}$
1000	3	31	9	$0,2 \times 10^{-1}$	$0,1 \times 10^{-1}$	$0,1 \times 10^{0}$	$0,3 \times 10^{-1}$
10000	7	181	41	0.2×10^{1}	$0,2 \times 10^{0}$	$0,46 \times 10^{1}$	$0,2 \times 10^{1}$
100000	8	164	15	0.2×10^{3}	$0,2 \times 10^{1}$	$0,5 \times 10^{2}$	0.4×10^{1}

Tabella 2.2: Nel prospetto la variabile *iter* è il numero di iterazioni che il solutore impiega nel tempo *time*, per raggiungere la tolleranza di 10^{-6} posta sul residuo *res*, con precondizionatore di Cholesky $\widehat{P}$, infine n è la dimensione della matrice incompleta A'.

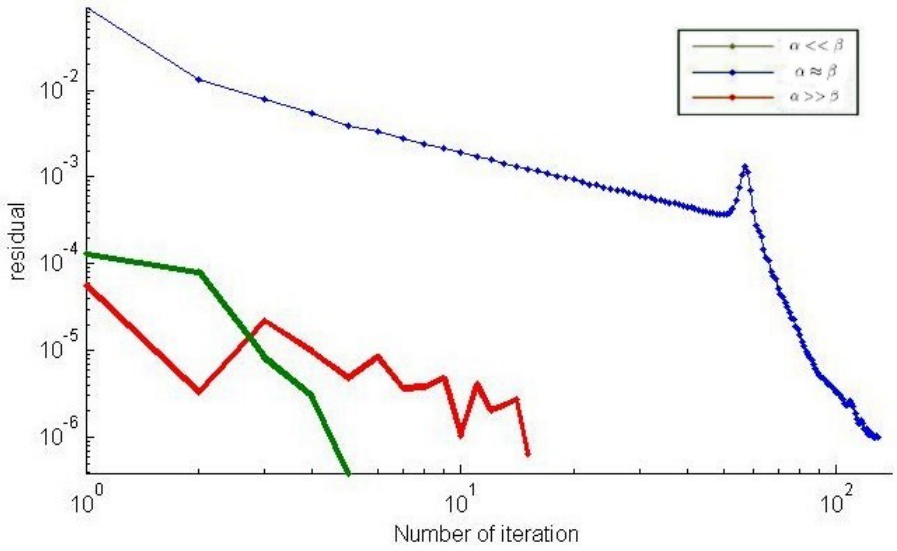

Figura 2.4: L' immagine mostra il comportamento del residuo relativo *res* al crescere del numero di iterazioni del PCG con precondizionatore di Cholesky $\widehat{P}$ quando $\alpha << \beta$ (verde) and $\alpha \approx \beta$ (blu) e $\alpha >> \beta$ (rosso) per $n = 100000$

A seguire si espongono in tabella (2.2) i risultati degli esperimenti quando il PCG usa il precondizionatore di Cholesky $\widehat{P}$, riportando il numero di iterazioni *iter* per giungere a tolleranza *tol*, il tempo di costruzione del precondizionatore di Cholesky $\widehat{P}$, indicato con $time_{\widehat{P}}$ e il tempo di risoluzione del sistema, indicato con $time_{\alpha<<\beta}$, $time_{\alpha\approx\beta}$ e $time_{\alpha>>\beta}$ rispettivamente se $\alpha << \beta$, $\alpha \approx \beta$ e $\alpha >> \beta$. Se si considera il caso in cui la dimensione è $n = 100000$ e $\alpha >> \beta$, le tabelle (2.1) e (2.2) mostrano che usando il precondizionatore diagonale il numero di iterazioni per raggiungere la tolleranza $tol = 10^{-6}$ è uguale a circa il $6,2\%$ della dimensione n del problema mentre usando il precondizionatore di Cholesky si necessitano solo di un numero di iterazioni pari a circa il 0.015% di n. Si consegue che il precondizionatore di Cholesky riduce il numero di iterazioni di un fattore di circa 413 volte

rispetto a quello diagonale. Tuttavia Il tempo totale di risoluzione di un sistema è dato dalla somma dei tempi per la costruzione del precondizionatore utilizzato e dal tempo della risoluzione del sistema precondizionato, che nel caso del precondizionatore P è di circa $0,7 \times 10^3$ s mentre per nel caso del precondizionatore $\widehat{P}$ è di circa $0,2 \times 10^2$ secondi. Relativamente al tempo di risoluzione totale abbiamo una riduzione di un fattore di solo $3,5$ volte, questo è dovuto alla complessità di calcolo della costruzione del precondizionatore di Cholesky che vanifica la ottenuta diminuzione del numero di iterazione. Inoltre sempre dalle tabelle (2.1) e (2.2) nel caso n=100000 si può notare che il tempo di esecuzione di una sola iterazione del PCG con precondizionatore diagonale è circa $0,11$ s mentre con precondiziontore di Cholesky è circa $0,26$ s cioè più del doppio. Questo fatto come già detto è dovuto alla risoluzione di un sistema lineare con matrice $\widehat{P}$ ad ogni iterazione del solutore per il precondizionamento del residuo $\mathbf{r}_k$. Infine riportiamo i risultati utilizzando il PCG con il precondizionatore inverso $\widehat{P}'$ in tabella (2.3).

Ricordiamo inoltre che questo precondizioantore è stato definito e costruito per superare il problema dello sbilanciamento dei coefficienti del nucleo ellittico quando $\alpha >> \beta$. Perciò osservando ancora il caso con $n = 100000$ e $\alpha >> \beta$, la riduzione nel numero di iterazione quando si passa dal precondizionatore diagonale P a quello inverso $\widehat{P}'$ è del circa il 50% mentre la riduzione temporale è di circa 25% rispetto il precondizionatore diagonale. Questa differenza tra riduzione del numero di iterazioni e del tempo di esecuzione è dovuto ancora una volta alla complessità di calcolo del matrice vettore $\widehat{P}'r_k$ per ogni iterazione del solutore rispetto a quella di Pr_k.

Infine concludiamo la sezione ribadendo come il tempo di esecuzione del PCG con il precondizionatore inverso $\widehat{P}'$ come anche con il precondionatore diagonale possono essere accelerati con l'uso di un opportuna strategia parallela del matrice vettore, per il calcolo $\widehat{P}'r_k$ e di Pr_k, su architetture di calcolo come quelle ibride CPU-GPU [10, 46, 18] al fine di diminuire il tempo di risoluzione del nucleo dinamico in (2.6) [44, 45]

n	$iter_{\alpha<<\beta}$	$iter_{\alpha\approx\beta}$	$iter_{\alpha>>\beta}$	$time_P$	$time_{\alpha<<\beta}$	$time_{\alpha\approx\beta}$	$time_{\alpha>>\beta}$
100	40	27	57	$0,2 \times 10^{-2}$	$0,19 \times 10^{-1}$	$0,13 \times 10^{-1}$	$0,27 \times 10^{-1}$
1000	100	73	576	$0,2 \times 10^{-1}$	$0,3 \times 10^{0}$	$0,23 \times 10^{0}$	$0,18 \times 10^{1}$
10000	735	444	2908	0.2×10^{0}	$0,15 \times 10^{2}$	$0,9 \times 10^{1}$	$0,61 \times 10^{2}$
100000	740	408	3184	0.2×10^{1}	$0,12 \times 10^{3}$	$0,68 \times 10^{2}$	0.53×10^{3}

Tabella 2.3: Nel prospetto la variabile *iter* è il numero di iterazioni che il solutore impiega nel tempo *time*, per raggiungere la tolleranza di 10^{-6} posta sul residuo res, con precondizionatore inverso $\widehat{P}'$, infine n è la dimensione della matrice incompleta A'.

2.7 Conclusioni

In questo capitolo si è evidenziato come, incrementare la rapidità di convergenza dei solutori dei nuclei dinamici nei modelli numerici oceanici, sia un obiettivo cruciale per la comunità oceanografica.

In principio si è numericamente mostrato come la rapidità di convergenza del solutore iterativo del nucleo ellittico in un modello oceanico dipenda principalmente dal rapporto dei suoi coefficienti α e β e dalla dimensione della griglia di discretizzazione sulla quale una sua approssimazione alle differenze finite è risolta. Nel particolare si è fatto vedere come l'attuale tecnica di precondizionamento utilizzata dal modello, rappresentata dal precondizionatore diagonale, possa risultare poco efficiente quando il dominio di risoluzione includa zone peculiari della Terra come i poli, al fine di descrivere quanto meglio la dinamica termoalina degli oceani. In seguito si è evidenziato come questa perdita di rapidità di convergenza sia dovuto ad un forte sbilanciamento dei coefficienti α e β del nucleo ellttico in esame.

Per superare questa problematica relativa allo sbilanciamento, si è proposto l'uso di una tecnica di precondizionamento di tipo inversa, basata sulla determinazione diretta di un'approssimazione dell'inversa del fattore incompleto di Cholesky, sfruttando le proprietà della matrice incompleta derivante dalla discretizzazione del nucleo in questione. La tecnica di precondizionamento scelta ha ridotto il tempo di esecuzione totale del 25% rispetto al tempo ottenuto utilizzando la tecnica di precondizionamento diagonale attuale nel caso dello sbilanciamento dei coefficienti del nucleo ellittico in disamina. La motivazione della scelta di una tecnica di precondizionamento di tipo in-

verso rispetta ad una di tipo ILU, per accelerare il solutore impiegato dal modello, è stato dettata principalmente in vista del futuro uso, anche da parte della comunità climatica, del nuovo paradigma di programmazione di "co-processing" tra CPU e GPU, per i motivi visti nel capitolo.

Capitolo 3

Il GPU Computing nella Risoluzione del Nucleo Ellittico di un Modello Oceanico

3.1 Introduzione

Al fine di accelerare i nuclei risolutivi, alla base dei modelli oceanici, anche da un punto di vista computazionale in questo capitolo si dimostra come l'utilizzo delle nuove architetture di calcolo parallelo GPU NVIDIA sono un' interessante opportunità per la comunità oceanografica, per migliorare la performance dei solutori iterativi nella risoluzione dei problemi oceanici. In questo capitolo il nucleo ellittico di OPA-NEMO è risolto mediante un' implementazione del PCG per mezzo dell'architettura di elaborazione in parallelo Computer Achitecture United Device (CUDA) e le libreria di cal-

colo scientifico Basic Linear Algebra Subprograms (BLAS) ottimizzata per il GPU computing.

In generale uno dei principali vincoli all'incremento delle risoluzioni delle griglie dei modelli oceanici è dato dall'aumento delle dimensioni dei problemi matematici derivanti, con il conseguente aumento delle operazioni matematiche richieste per la determinazione della soluzioni di questi. Inoltre lo stesso problema matematico risolto su parti differenti del dominio globale di questi modelli oceanici, la Terra, può avere un costo di risoluzione che può variare considerevolmente. Un esempio di questo fatto avviene molto spesso nelle zone in prossimità dei poli terrestri. Tuttavia sebbene la superficie e il volume degli oceani polari Artico ed Antartico sono trascurabili quando comparati all'dominio globale, le loro dinamiche hanno un'elevata influenza sulla circolazione oceanica globale. Questo è dovuto ai processi di fusione e di congelamento che interessano queste zone, le quali comportano un elevato scambio di energia. Inoltre esse costituiscono una grande riserva di acqua e di sale, e quindi sono interessate da un elevato scambio di materia. Questi fatti portano a prendere in seria considerazione i poli della Terra in un modello oceanico globale. L'obiettivo qui è mostrare come il GPU computing può inserirsi nella strategia parallela dei modelli oceani e in particolare in quella del modello oceanico globale OPA-NEMO per superare i problemi prima citati.

Le GPU rappresentano le nuove architetture di calcolo parallelo utili per accelerare la risoluzione dei flussi di operazioni più onerosi su processore se quest'ultimo è dotato di una scheda grafica NVIDIA. L'utilizzo del iperpar-

allelismo delle GPU su calcolatore può essere la soluzione anche perché, per avere buone prestazioni rispetto le Control Processor Unit (CPU), le dimensioni dei problemi matematici devono appunto crescere. Nel particolare qui si mostrerá come il GPU computing applicato al PCG , prima con precondizionatore diagonale e poi con precondizionatori di tipo inverso, può apportare un considerevole aumento della performance temporale del nucleo ellittico del modello oceanico OPA-NEMO.

I risultati ottenuti nel capitolo vogliono essere una proposta all'uso da parte della comunità oceanografica delle nuove rivoluzionarie architetture di calcolo GPU combinate con tecniche di precondizionamento più efficienti con le quali possono accelerare enormemente i solutori iterativi di questi modelli oceanici dando la possibilità di andare oltre nelle simulazioni oceaniche.

3.2 Approfondimenti sul Nucleo Ellittico in OPA-NEMO

Il nucleo ellittico del modello oceanico OPA-NEMO è rappresentato, come già osservato, dal seguente problema di Laplace:

$$\begin{cases} \dfrac{\partial}{\partial i}\left[\dfrac{e_2}{He_1}\dfrac{\partial}{\partial i}\left(\dfrac{\partial \psi}{\partial t}\right)\right] + \dfrac{\partial}{\partial j}\left[\dfrac{e_1}{He_2}\dfrac{\partial}{\partial j}\left(\dfrac{\partial \psi}{\partial t}\right)\right] = \\ \qquad\qquad \dfrac{\partial}{\partial i}(e_2\bar{M}_v) - \dfrac{\partial}{\partial j}(e_1\bar{M}_u) \;\; su\;\Omega \times [t_0\; t_1] \\ \dfrac{\partial \psi}{\partial t} = 0 \quad su\;\delta\Omega \times [t_0\; t_1] \end{cases} \tag{3.1}$$

La funzione $\frac{\partial \psi}{\partial t}$ (con ψ chiamata *Volume Transport Stream function*) è l'incognita del problema (3.1). Nel problema di Laplace (3.1) i termini e_1, e_2 e e_3 [87] rappresentano i fattori di scala quando si passa da un sistema di coordinate cartesiane (x, y, z) a un sistema di coordinate curvilinee (i, j, k), ad esempio quelle sferiche essendo il dominio del modello oceanico in esame l'intero pianeta Terra. Inoltre la funzione $H(i, j)$ è la superficie che parametrizza il fondo oceanico ed infine $\bar{M} = (\bar{M}_u, \bar{M}_v)$ rappresentano i contributi dell'accelerazione di Coriolis, della pressione idrostatica e i termini di avvezione integrati lungo l'asse k con direzione del raggio terrestre. La funzione $\frac{\partial \psi}{\partial t}$ è necessaria al modello oceanico per il calcolo del gradiente di pressione superficiale $\nabla_h p_s$ nelle equazioni di Navier Stokes in OPA-NEMO [74]:

$$\frac{\partial \mathbf{U}_h}{\partial t} = -\left[(\nabla \times \mathbf{U}) \times \mathbf{U} + \frac{1}{2}\nabla(\mathbf{U}^2)\right]_h - f\mathbf{k} \times \mathbf{U}_h - \frac{1}{\rho_0}\nabla_h(p_i + p_s) + \mathbf{D}^{\mathbf{U}} \quad (3.2)$$

e si introduce nelle equazioni (3.2) mediante la seguente uguaglianza:

$$\frac{1}{\varrho_0}\nabla_h p_s = M - \frac{1}{H}\left[\vec{k} \times \nabla\left(\frac{\partial \psi}{\partial t}\right)\right]. \quad (3.3)$$

Nelle equazioni (3.2) i vettori $\mathbf{U}$ e $\mathbf{U}_h$ rappresentano rispettivamente il campo tridimensionale e bidimensionale orizzontale delle velocità, p_i e p_s sono rispettivamente la pressione idrostatica e la pressione superficiale, f è l'accelerazione di Coriolis, ϱ_0 è la densità media degli oceani ed infine D^U è una quantità che parametrizza fenomeni di piccola scala legati al momento.
Posto con $\alpha(i, j) = \frac{e_2}{He_1}$ e con $\beta(i, j) = \frac{e_1}{He_2}$, i coefficienti di Laplace del problema (3.1), essi hanno su Ω sempre lo stesso segno per la condizione di ellitticitá e inoltre appartengano a $C^1(\bar{\Omega})$. Se si utilizza il sistema curvilineo sferico (λ, ϕ, r) (rispettivamente note come longitudine, latitudine e raggio)

per la rappresentazione del problema (3.1) essendo Ω una sfera, allora i fattori di scala e_1, e_2 e e_3 rispetto a λ, ϕ e r sono i seguenti:

$$e_1 = r\cos\phi, \quad e_2 = r, \quad e_3 = 1, \ con\ \phi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \tag{3.4}$$

In prossimità del polo nord $(\lambda, +\frac{\pi}{2}, r)$ e del polo sud $(\lambda, -\frac{\pi}{2}, r)$, per i coefficienti di Laplace, che in coordinate sferiche diventano $\alpha(\phi) = \dfrac{1}{H\cos\phi}$ e $\beta(\phi) = \dfrac{\cos\phi}{H}$, valgono rispettivamente i seguenti limiti:

$$\begin{aligned} \forall M > 0,\ \exists \delta_M > 0\ :\ \forall \phi \in \left[\frac{\pi}{2} - \delta_M,\ \frac{\pi}{2}\right] \Longrightarrow \alpha(\phi) > M. \\ \forall \epsilon > 0,\ \exists \delta_\epsilon > 0\ :\ \forall \phi \in \left[\frac{\pi}{2} - \delta_\epsilon,\ \frac{\pi}{2}\right] \Longrightarrow 0 < \beta(\phi) < \epsilon \end{aligned} \tag{3.5}$$

e

$$\begin{aligned} \forall M > 0\ \exists \delta_M > 0\ :\ \forall \phi \in \left[-\frac{\pi}{2},\ -\frac{\pi}{2} + \delta_M\right] \Longrightarrow \alpha(\phi) > M. \\ \forall \epsilon > 0\ \exists \delta_\epsilon > 0\ :\ \forall \phi \in \left[-\frac{\pi}{2},\ -\frac{\pi}{2} + \delta_\epsilon\right] \Longrightarrow 0 < \beta(\phi) < \epsilon \end{aligned} \tag{3.6}$$

Da cui segue che se scegliamo $M >> \epsilon$ esisteranno degli intervalli del tipo $\left[\frac{\pi}{2} - \delta,\ \frac{\pi}{2}\right]$ o $\left[-\frac{\pi}{2},\ -\frac{\pi}{2} + \delta\right]$ con $\delta = \min(\delta_\epsilon, \delta_M)$ per cui vale la seguente diseguaglianza:

$$\alpha(\phi) > M >> \epsilon > \beta(\phi) \tag{3.7}$$

Quindi in prossimità dei poli si hanno diversi ordini di grandezza di differenza tra i coefficienti di Laplace ovvero $\beta/\alpha << 1$. In prossimità dell' equatore, essendo $\phi \approx 0$, il $\cos(\phi)$ è all'incirca 1, pertanto vale la seguente:

$$\frac{1}{H\cos(\phi)} \approx \frac{\cos\phi}{H} \tag{3.8}$$

ovvero $\alpha \approx \beta$.

I risultati (3.5),(3.6) e (3.7) in prossimità dei poli vanno ad incidere sulla

velocità di convergenza del solutore iterativo del modello, come visto nel paragrafo (2.4) del capitolo precedente.

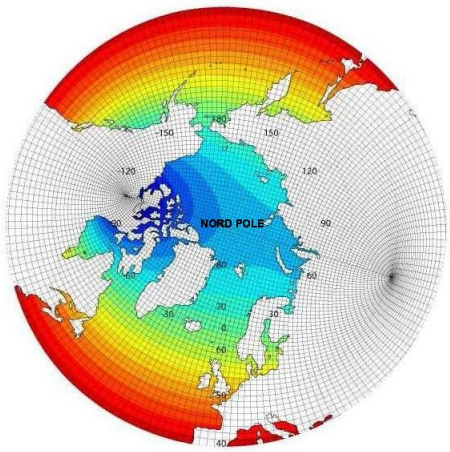

Figura 3.1: Parte della rappresentazione del dominio terrestre di OPA-NEMO

Sempre nel capitolo (2) si è visto che dato il dominio Ω ed una griglia finita $\Omega_{k,h}$ con $2p \times 2q$ punti di Ω, mediante le differenze finite del secondo ordine centrate, applicate ad (3.1) ad ogni istante $t_m \in [t_0, t_N]$, si passa al seguente problema numerico:

$$\begin{cases} h^2\left(\alpha(i+k,j)\frac{\partial\psi(i+2k,j)}{\partial t} - \big(\alpha(i+k,j)+\alpha(i-k,j)\big)\frac{\partial\psi(i,j)}{\partial t} + \alpha(i-k,j)\frac{\partial\psi(i-2k,j)}{\partial t}\right) + \\ +k^2\left(\beta(i,j+h)\frac{\partial\psi(i,j+2h)}{\partial t} - \big(\beta(i,j+h)+\beta(i,j-h)\big)\frac{\partial\psi(i,j)}{\partial t} + \beta(i,j-h)\frac{\partial\psi(i,j-2h)}{\partial t}\right) \\ \qquad\qquad = (2hk)^2 f(i,j,t_m) \quad su\ \Omega_{k,h} \\ \\ \frac{\partial\psi(i,j)}{\partial t} = 0 \quad su\ \delta\Omega_{k,h} \end{cases} \tag{3.9}$$

Nel problema (3.9) k e h sono i passi di discretizzazione rispettivamente lungo gli assi i e j. Si è inoltre osservato che per determinare la soluzione di (3.9) ad ogni passo temporale $t_m = t_0 + \frac{m}{N}(t_N - t_0),\ m = 1, ..., N$ del modello oceanico si deve risolvere un sistema lineare con matrice incompleta A di dimensione $n = p \times q$ sparsa, simmetrica e definita positiva [95]. La sparsità di A è rappresentata in figura (3.2) e gli elementi sulle sue diagonali sono dati dalle seguenti:

$$\begin{aligned} a_{m,m} &= h^2\big(\alpha(i+1,j) + \alpha(i-1,j)\big) + k^2\big(\beta(i,j+1) + \beta(i,j-1)\big) \\ a(m,m-1) &= -h^2\alpha(i-k,j), \quad a(m,m+1) = -h^2\alpha(i+k,j) \\ a(m,m-q) &= -k^2\beta(i,j-h), \quad a(m,m+q) = -k^2\beta(i,j+h) \end{aligned} \tag{3.10}$$

dove $m = 1, ..., n$ è legato agli indici del dominio discreto secondo la seguente relazione $m = (\mathtt{i} - 1)q + \mathtt{j}$ (con $\mathtt{i} = 1, ..., p$ and $\mathtt{j} = 1, ..., q$).

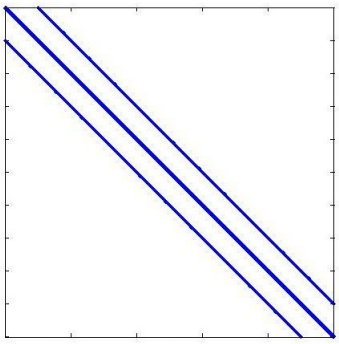

Figura 3.2: Sparsità della matrice A

Il problema (3.9), ad ogni passo temporale del modello oceanico t_m, si riduce nella risoluzione di un sistema lineare indicato con:

$$Ax = b \tag{3.11}$$

Essendo A una matrice con un elevato indice di sparsità $\big(\texttt{SP}(A) < 1 - \frac{5}{n}\big)$, si è detto che OPA-NEMO utilizza un metodo di risoluzione per sistemi di equazioni algebriche di tipo iterativo per (3.11), sfruttando la sparsità nell'implementazione del matrice vettore, operazione dominante nei metodi iterativi. Inoltre essendo A simmetrica e definita positiva, tra i metodi iterativi noti in letteratura [5], OPA-NEMO utilizza il PCG con precondizionatore P diagonale. Il PCG determina una successione approssimante $\{x_m\},\ m \in \mathbb{N}$ della soluzione di (3.11) nel seguente spazio di Krylov:

$$x_m \in span\big\{x_0, P^{-\frac{1}{2}}AP^{-\frac{1}{2}}x_0, (P^{-\frac{1}{2}}AP^{-\frac{1}{2}})^2x_0, ..., (P^{-\frac{1}{2}}AP^{-\frac{1}{2}})^m x_0\big\}. \tag{3.12}$$

Inoltre ad ogni passo l'errore $e_m = x - x_m$ appartiene allo spazio:

$$e_0 + span\big\{P^{-\frac{1}{2}}AP^{-\frac{1}{2}}e_0, (P^{-\frac{1}{2}}AP^{-\frac{1}{2}})^2e_0, ..., (P^{-\frac{1}{2}}AP^{-\frac{1}{2}})^m e_0\big\} \tag{3.13}$$

con $e_0 = x - x_0$. Tra tutti i vettori dello spazio in (3.13), si dimostra che e_m è quello che ha la $A-$norma minima e che la successione $\{x_m\},\ m \in \mathbb{N}$ converge in al più n passi [90] dove n è la dimensione di (3.11). Il PCG si basa basa su un metodo iterativo di tipo *Simple Iteration* [51]:

$$x_m = x_{m-1} - P^{-1}(b - Ax_{m-1}) \tag{3.14}$$

Al passo m l'errore $e_m = x - x_m$ è dato da:

$$\begin{aligned} x - x_m &= x - x_{m-1} - P^{-1}(b - Ax_{m-1}) \\ e_m &= e_{m-1} - P^{-1}Ae_{m-1} \iff \\ e_m &= (I - P^{-1}A)e_{m-1} \end{aligned} \tag{3.15}$$

Segue che una stima della rapidità di convergenza del PCG è data da:

$$\|e_m\| \leq \|(I - P^{-1}A)\|\ \|e_{m-1}\| \tag{3.16}$$

da cui segue che:

$$\|e_m\| \leq \|(I - P^{-1}A)\|^m \ \|e_0\| \tag{3.17}$$

Utilizzando le diseguaglianze (2.20), (2.21) e (2.22) del capitolo precedente, le conclusioni sono le seguenti: aumentare le griglie di risoluzione al fine di migliorare l'accuratezza delle simulazioni dei modelli oceanografici o risolvere tali modelli su alcuni porzioni del dominio, nel caso specifico i poli terrestri, comportano (3.11) una diminuzione della rapidità di convergenza del solutore iterativo scelto per la risoluzione del sistema (3.9) con un conseguente aumento del costo computazionale a carico delle CPU. Al fine di superare queste barriere si propone di utilizzare una architettura di computing Multiple Instruction Multiple Data (MIMD) formata da sistemi ibridi con GPU a core paralleli che lavorano in tandem con CPU multi-core.

3.3 Risoluzione del Problema mediante PCG su GPU

In questa sezione, si descrive un'implementazione del PCG per il sistema lineare strutturato (3.11), che può essere eseguito su qualsiasi GPU NVIDIA che supporta CUDA. Uno degli svantaggi delle GPU è il profondo divario di prestazione tra la doppia e singola precisione. Si pensi che per le più moderne GPU e solo per i chip basati su architettura con *compute capability* a partire da 1.3, una singola unità Streaming Processor (SP) su 8 è a doppia precisione (64 bit). Da questo fattore, è evidente che le prestazioni massime a 64 bit saranno al più un ottavo rispetto a quelle a 32 bit, supposto di pot-

er sfruttare appieno le capacità computazionali dell'architettura. Per questi motivi, presentiamo una versione del PCG in singola precisione, tratto da [44], sviluppato mediante le librerie CUDA e CUDA Basic Linear Algebra Subprograms (CUBLAS) e paragonato ad una versione sequenziale fondata sul linguaggio C e la libreria CBLAS. L'algoritmo del PCG, il cui pseudo codice è tratto da [34], è mostrato nello schema (1) del capitolo precedente. La sua operazione più dispendiosa, dal punto di vista della complessità di calcolo è il prodotto matrice per vettore $\mathbf{y} = A\mathbf{d}_k$, eseguito ad ogni iterazione del solver. Il costo di questa operazione algebrica è un $O(n^2)$ se è eseguito senza utilizzare un formato di memorizzazione sparso per A.

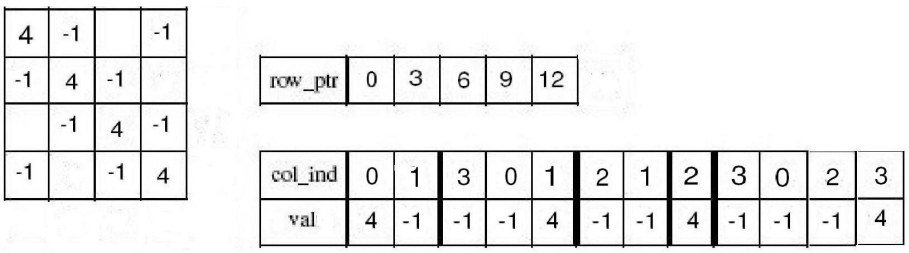

Figura 3.3: Schema di memorizzazione di una matrice sparsa in CRS

Usando un opportuno schema di memorizzazione per A, come il formato Compressed Row Storage (CRS), ed un opportuna implementazione del prodotto $\mathbf{y} = A\mathbf{d}_k$, adattato al CRS, la complessità di calcolo dell' operazione matriciale cala fino ad un $O(n)$. Il CRS è uno tra gli schemi di memorizzazione più utilizzati per le matrici sparse. Questo schema di memorizzazione utilizza tre vettori, indicati *val*, *col_index* e *row_ptr* per memorizzare A come mostrato in figura (3.3). Il vettore *val* di tipo float o double memorizza per righe gli elementi consecutivi di A diversi da zero. Il vettore *col_index* memo-

rizza l'indice di colonna degli elementi contenuti in val ovvero se $val(k) = a_{i,j}$ allora $col\_index(k) = j$. Infine il vettore row_ptr memorizza nella sua posizione i il posto in val del primo elemento diverso da zero della $i-$esima riga di A ovvero se $val(k) = a_{i,j}$, allora $row\_ptr(i) \leq k < row\_ptr(i+1)$. Per convenzione si pone $row\_ptr(n+1) = nz$, dove nz è il numero di elementi diversi da zero della matrice A. Con il formato CRS al posto di utilizzare n^2 locazioni di memoria per la matrice A, se ne utilizzano solo $2nz + n + 1$. Un algoritmo che calcola il prodotto matrice per vettore adattato a questo formato è descritto nello schema (5). Il tipo di parallelismo implementato

Schema 5 Algoritmo del matrice $\times$ vettore $Ad = y$ nel formato CRS

for $i \leftarrow 0$ **to** $n-1$ **do**
  $y(i) \leftarrow 0$
  for $j \leftarrow row\_ptr(i)$ **to** $row\_ptr(i+1) - 1$ **do**
    $y(i) \leftarrow y(i) + val[j] \times d(col\_ind(j))$
  end for
end for

in CUDA è di tipo Single Instruction Multiple Threads (SIMT), dove una moltitudine di thread differenti eseguono congiuntamente la stessa istruzione su dati differenti. Un algoritmo efficiente in CUDA che calcoli il prodotto matrice per vettore $\mathbf{y} = A\mathbf{d}_k$ è tale che l'i-esima riga ($i \leq n$) di A sia assegnata all' i-esimo thread dell'architettura (figura 3.4); quest'ultimo calcola il prodotto puntuale tra la riga attribuitagli e il vettore $\mathbf{d}_k$ al fine di ottenere la i-esima componente del vettore $\mathbf{y}$. Il Kernel in CUDA che implementa il prodotto matrice per vettore nel formato CRS in modo efficiente è rappre-

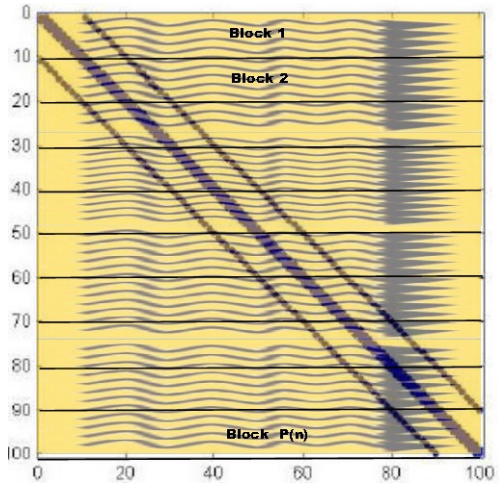

Figura 3.4: Distribuzione della matrice A tra i thread della GPU

sentato in schema (6) [9].

Schema 6 Kernel che implementa il Matrice per Vettore nel formato CRS

```
__global__ void spmv_crs_scalar_kernel( int n , int* row_ptr ,const int*
    col_ind , const float * values ,const float* d , float* y)
  {
     int i = blockDim.x * blockIdx.x + threadIdx.x ;
      if ( i < n )
     {
       int  pr1=row_ptr[i];
       int  pr2=row_ptr[i+1]-1;
       y[i]=0;
       for (int j =pr1; j<=pr2; j++)
           y[i]=y[i]+val[j]*d[col_ind[j]-1];
     }
  }
```

Il precondizionamento del sistema lineare $A\mathbf{x} = \mathbf{b}$ mediante precondizionatore diagonale P è realizzato all'interno del algoritmo del PCG mediante l'istruzione $\widehat{\mathbf{r}}_k = P^{-1}\mathbf{r}_k$ (schema (1)) ad ogni iterazione del solver. Essendo $P = diag(A)$ una matrice diagonale allora $P^{-1} = (diag(A))^{-1}$ e dunque $\widehat{r}_k(i) = \left(\dfrac{r_k(i)}{a(i)^{-1}}\right),\ i = 1, ..., n$. Se si memorizza P in un array monodimen-

sionale e si impone che l'i-esimo thread della struttura calcoli l'i-esima componente $\widehat{r}_k(i)$ di $\widehat{\mathbf{r}}_k$ allora il codice CUDA che effettua ciò è dato in schema (7).

Per le altre operazioni del solver PCG come il prodotto puntuale tra due

Schema 7 //Algoritmo che calcola il prodotto tra le componenti dei vettori pinv e r

```
__global__ void vecProdKernel(float* pinv, float* r, float* z)
{
        int i = blockDim.x*blockIdx.x+ threadIdx.x;
        z[i] = pinv[i] * r[i];// prodotto tra le componenti di pinv e r
}
```

vettori, il prodotto tra uno scalare per un vettore, la saxpy e il calcolo della norma di un vettore si è usata la libreria CUBLAS che è la libreria di algebra lineare ottimizzata per CUDA introdotta nella sezione (A.12) di questo capitolo. Nello schema (8) mostriamo il nucleo del software PCG su GPU: Per sfruttare a pieno la potenza dell'architettura grafica, la progettazione del kernel deve tener presente alcuni dettagli tecnici quali il numero massimo di thread per block, il numero massimo di thread per ogni Stream Multiprocessor (SM) ed infine il numero massimo blocchi per SM. Con una scheda grafica con compute capability 2.0, che è quella utilizzata per i nostri esperimenti, questi valori sono definiti nel seguente modo: il massimo numero di thread per blocco è 512, il massimo numero di thread per SM è 1536 e il massimo numero di blocchi per SM è 8. Da questi dati avremo che ad ogni SM potranno essere assegnati un massimo di $1536/32 = 48$ warp. Quindi per riempire la capacità elaborativa dello Streaming Multiprocessor si ha che innanzi tutto il numero di thread per block deve essere divisibile per 32, altrimenti avremmo

Schema 8 Nucleo dell'Implemetazione del PCG su GPU

```
\\ Criterio di arresto posto sul residuo e sul massimo di numero di iterazioni al più
\\ pari alla dimensione del problema.
while(ratio_norme>TOL && iterMax>ct)
 {
  //Contatore del numero di iterazioni
  ct++;
  //Chiamata al Kernel che implementa il matrice vettore sparso sulla device y=Ad
  //nel formato di memorizzazione CRS
  spmv_csr_kernel<<<GridDim,BlockDim>>>(n,d_row_ptr,d_col_index,d_val,d_d, d_y);
  //Blocca la CPU fino al completamento del kernel sulla GPU
  cudaThreadSynchronize();
  //Controlla se l'esecuzione del kernel ha generato qualche errore
  checkCUDAError("kernel␣execution");
  //Calcola il prodotto scalare a_den=(d,y) sulla GPU
  a_den=cublasSdot(n,d_d,incd,d_y,incy);
  //Calcola il prodotto scalare a_num=(z,r) sulla GPU
  a_num=cublasSdot(n,d_z,incz,d_r,incr);
  //Calcolo del parametro alpha_k sulla CPU per l'aggiornamento della soluzione x_k+1
  a_alpha=a_num/a_den;
  //Aggiornamento della soluzione x_{k+1}=x_{k}+ alpha_k d
  cublasSaxpy(n,a_alfa,d_d,incd,d_x,incx);
  //Aggiornamento del residuo r_{k+1}=r_{k}-alpha_k y
  cublasSaxpy(n,-a_alfa,d_y,incy,d_r,incr);
  //Kernel per il precondizionamento di Ax=b mediante z_k+1=P^-1 r_k+1
  vecProdKernel<<<GridDim,BlockDim>>>(d_r, d_precond, d_z);
  //Blocca la CPU fino al completamento del kernel sulla device
  cudaThreadSynchronize();
  //Controlla se l'esecuzione del kernel ha generato qualche errore
  checkCUDAError("kernel␣execution");
  // Calcolo del numeratore del parametro beta mediante prodotto scalere
  // b_num=(z_k+1,d_k+1)
  b_num=cublasSdot(n,d_z,incz,d_r,incr);
  //Calcolo del parametro beta per l'aggiornamento della nuova direzione di ricerca
  //della soluzione
  b_beta=b_num/a_num;
  //Calcolo del prodotto scalare vettore beta*d
  cublasSscal(n,b_beta,d_d,incd);
  //Calcolo della nuova direzione di ricerca d_{k+1}=z_k+1+ beta_k d_k
  cublasSaxpy(n,alfa,d_z,incz,d_d,incd);
  //Calcolo della norma 2 del residuo
  norma1=cublasSnrm2(n,d_r,incr);
  //Calcolo del della norma del residuo relativo al vettore dei termini noti b
  ratio_norme=norma1/norma2;
   }
```

l'assegnazione di warp parzialmente riempiti (verranno infatti aggiunti dei thread fittizi per raggiungere i 32 necessari), inoltre il numero di warp per block deve essere un divisore di 48, altrimenti non potrà essere raggiunto il numero massimo di warp assegnabili ad ogni SM. Sempre per riempire lo SM, considerando il vincolo del numero massimo di blocchi assegnabili alla SM, ogni blocco dovrebbe avere $48/8 = 6$ warp, quindi 192 thread per blocco. Adoperando queste criteri, una configurazione di lancio del Kernel per massimizzare l'utilizzazione della device è la mostrato in schema (9):

Schema 9 Configurazione del lancio del Kernel

```
const int WarpSize = 32;
const int maxGridSize = 65535;//Maximum number of simultaneous Blocks
int warpCount = (n / warpSize) + (((n % warpSize) == 0) ? 0 : 1);
int warpPerBlock = max(1, min(6, warpCount));
int threadCount = warpSize * warpPerBlock;
int blockCount = min( maxGridSize, max(1, warpCount/warpPerBlock) );
if (n>threadCount*blockCount) blockCount++;
dim3 BlockDim = dim3(threadCount, 1, 1);
dim3 GridDim = dim3(blockCount, 1, 1);
```

3.4 Precondizionatori Inversi nel Modello Oceanico su GPU

I più comuni precondizionatori algebrici possono suddividersi in due grandi categorie rappresentati da quelli di tipo ILU e Sparse Approximate Inverses (SAINV) [12]. I precondizionatori del tipo ILU, come la fattorizzazione incompleta di Cholesky, sono dei potenti precondizionatori capaci di incrementare enormemente la velocità di convergenza dei solutori iterativi. E' pero noto che il peggiore svantaggio per l'utilizzo degli ILU per problemi di

grandi dimensioni è la difficoltà nella parallelizzazione. Infatti questi ultimi soffrono per lo più della forward e della backward substitution richieste per la loro applicazione nel algoritmo del PCG. I precondizionatori P' appartenenti alla seconda categoria invece sono concettualmente già paralleli poiché provvedono ad una esplicita approssimazione di A^{-1} [57, 7]. Infatti loro applicazione nel PCG necessita solo di un matrice per vettore sparso per il calcolo di $\widehat{\mathbf{r}}_k = P'\mathbf{r}_k \quad k = 1, ..., n$ e quindi è facilmente parallelizzabile soprattutto su architetture, quali, le GPU. In letteratura il problema della costruzione di un'approssimazione dell' inversa ha due approcci tipici, il primo basato su un problema di ortogonalizzazione e il secondo su un problema di minimizzazione. Il primo è fondato sull'algoritmo incompleto di Gram Schmidt che genera un'approssimazione della fattorizzazione triangolare di A^{-1} considerando solo gli elementi diversi da zero di A. Questa procedura è la base del noto algoritmo denominato AINV [15, 14] e il precondizonatore ottenuto dá ai solutori iterativi una grande rapidità di convergenza. Tuttavia lo svantaggio dei precondizionatori AINV sta nel fatto che la loro costruzione è per lo più sequenziale. L'altra classe di precondizionatori inversi è fondata su un problema di minimizzazione nella norma di Frobenius della distanza $\|P^{-1}A - I\|_F$ che può essere ottenuta mediante la soluzione di n problemi indipendenti ai minimi quadrati soggetti ad alcuni vincoli di sparsità [57]. In un ambiente di calcolo distribuito, ogni processore è in grado di acquisire i coefficienti della matrice necessari per risolvere il suo problema ai minimi quadrati. Quindi, l'algoritmo corrispondente può essere implementato efficientemente su una macchina parallela, perché ogni processore può es-

eguire la costruzione della propria parte di precondizionatore senza overhead di comunicazione. Tuttavia il precondizionamento apportato da quest'ultimi è meno efficiente di quello della prima classe. Tra le due tipologie di precondizionatori inversi, utilizzeremo dei precondizionatori di tipo AINV per il nostro problema in quanto la principale difficolta è la risoluzione di più sistemi lineari

$$Ax = b_m \quad m = 1, ..., N. \tag{3.18}$$

con la stessa matrice dei coefficienti A. Il precondizionatore nel nostro caso verrà dunque costruito un' unica volta perciò a noi interessa soprattutto accelerare la rapidità di risoluzione dei sistemi in (3.18).

L' idea alla base dei precondizionatori $AINV$ per matrici simmetriche e definite positive, come la matrice A, è la fattorizzazione

$$A = LDL^T \Longrightarrow \quad A^{-1} = (L^{-1})^\top D^{-1} L^{-1} \tag{3.19}$$

con L matrice unitaria triangolare inferiore, ottenuta mediante fattorizzazione LU e $D = diag(U)$ matrice diagonale, i cui elementi della diagonale principale coincidono con i corrispondenti elementi della matrice triangolare superiore della fattorizzazione LU di A, segue che $L^\top = D^{-1}U$.

Essendo $(L^{-1})^\top$ generalmente una matrice densa triangolare superiore, la sparsità è preservata riducendo il *fill in* durante il calcolo dei vettori colonna di $(L^{-1})^\top$ al di sopra della diagonale principale. Questa idea di calcolare un' approssimazione sparsa di $(L^{-1})^\top$ per costruire un precondizionatore per il PCG fu proposto prima in [11] e poi [13, 48, 15]

Questo può essere ottenuto o eliminando il *fill in* in certe posizioni della matrice come fatto nel capitolo precedente nel caso dell' inversa del fattore incompleto di Cholesky U^{-1} per la costruzione di $\widehat{P}'$ oppure trascurando gli elementi di $(L^{-1})^{\top}$ al di sotto di una tolleranza prestabilita [13].

In generale le motivazioni di questi approcci sono basata su risultati teorici e esperimenti al calcolatore che mostrano come molti degli elementi dell'inversa di una matrice A sparsa, simmetrica e definita positiva siano in norma molto piccoli [33, 78].

Al fine di accelerare la rapidità di convergenza del PCG su GPU abbiamo dunque sostituito il precondizionatore diagonale con i precondizionatori di tipo AINV realizzati dalla libreria scientifica per il calcolo parallelo per matrici sparse CUSP su GPU [32] e il precondizionatore $\widehat{P}'$.

Le loro prestazioni paragonate al precondizionatore diagonale in parallelo sono mostrate nella sezione degli esperimenti numerici (3.5).

3.5 Esperimenti numerici

Il modello numerico (3.9) è risolto, come detto nel paragrafo (3.2) di questo capitolo, ad ogni passo temporale $t_m = t_0 + \frac{m}{N}(t_N - t_0)$ $(m = 1, ..., N)$ del modello oceanico OPA-NEMO. Necessariamente dovranno essere risolti N sistemi lineari del tipo $Ax = b_m$ $(m = 1, ..., N)$. *ORCA025* [41, 43] è uno dei principali software che implementa il modello OPA-NEMO alla configurazione di $\frac{1}{4}^{\circ}$ di grado e fa uso di una griglia di risoluzione $\Omega_{k,h}$ con 1440×720 punti, ne consegue che la dimensione della matrice A sarà del ordine di $O(10^6)$. Dovuto ai grossi costi computazionali per la risoluzione dei sistemi lineari

$Ax = b_m$, si è implementato il PCG su GPU mediante l'architettura di calcolo parallelo CUDA e la libreria di calcolo scientifico CUBLAS per GPU. In questo paragrafo si paragona principalmente la versione parallela ideata su GPU con una classica sequenziale per processori CPU, realizzata mediante il linguaggio C e CBLAS, sulla base della performance. L'architettura di calcolo utilizzata per la sperimentazione ha due schede madri, ognuna delle quali capace di supportare due CPU. Nello specifico, l'architettura consiste di due server, ognuno delle quali lavora con 2 CPU "Intel Xeon E5620" da 2.4 GHz quad-core, con 12 MB di memoria cache. Inoltre, essa è dotata di una scheda grafica "NVIDIA TESLA S2050", la quale è dotata di 4 GPU, ognuna con 448 core con velocità 1.55 GHz per un totale 1792 core. La scheda grafica"TESLA S2050" è basato sulla nuova tecnologia "FERMI GPU" [82], il cui schema è riportato a destra di figura (3.5). La Tesla in questione può anche essere usata con host che hanno una sola slot PCI Exspress, in questo modo, ogni sistema host potrà accedere a due delle quattro GPU come mostrato a sinistra della figura (3.5) Gli esperimenti numerici, condotti in singola

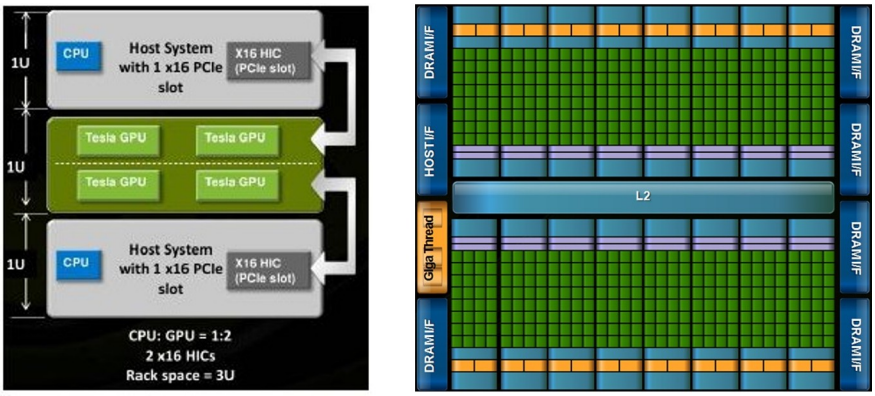

Figura 3.5: A sinistra schema dell'architettura di calcolo utilizzata a destra modello della nuova scheda grafica Fermi

precisione, per la risoluzione del sistema lineare $Ax = b$ sono stati realizzati nel seguente modo: si è posto x=(1,...,1) e calcolato $y = Ax$, poi si è fissato $b = y$ e infine si sono dati A e b in input al PCG sequenziale (su CPU) e parallelo (su GPU) con precondizionatore diagonale. ottenendo come output il numero di iterazioni essenziale per ottenere una tolleranza prestabilita, la soluzione e le accuratezze calcolate in norma euclidea e in norma infinito. Inoltre per la risoluzione di $Ax = b$ si è incrementato la dimensione n del sistema lineare, nell'intervallo di estremi $n = 1000$ e $n = 100000$, di un fattore di 10 e variando il rapporto tra le funzioni α e β sul dominio. La figura (3.6) esprime il significato fisico degli esperimenti dove si vuole sottoliniare il legame tra la dimensione del sistema alla risoluzione delle griglie spaziali del modello e la dipendenza degli elementi della matrice dei coefficienti A al sottodominio del modello che si vuole simulare.

Gli esperimenti, i cui risultati soni riportati nelle tabelle di seguito, sono

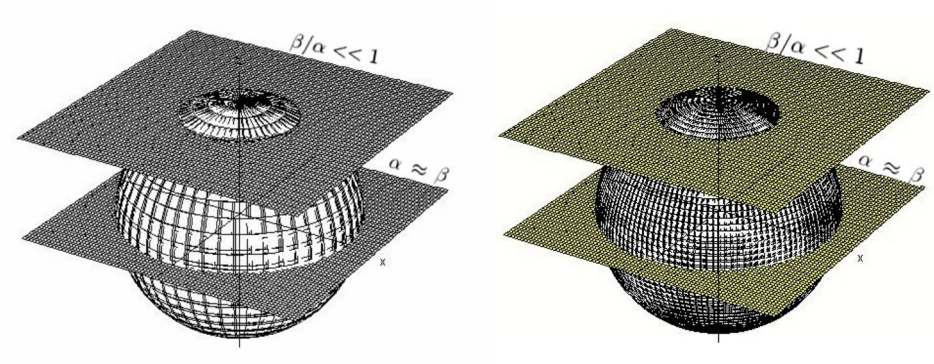

Figura 3.6: La figura rappresenta due sfere con risoluzioni di griglia differenti sulla quale il rapporto dei coefficienti di Laplace varia come indicato

stati effettuati fissando nello specifico una tolleranza di 10^{-6}. L'algoritmo

n	**Versione**	**iter**$_{\alpha<<\beta}$	**iter**$_{\alpha\approx\beta}$	**iter**$_{\alpha>>\beta}$
1000	Sequenziale CPU	56	83	564
	Parallela GPU	56	83	564
10000	Sequenziale CPU	501	402	7549
	Parallela GPU	501	402	7549
100000	Sequenzale CPU	499	654	23334
	Parallela GPU	499	654	23334

Tabella 3.1: La tabella mostra il numero di iterazioni iter necessarie per raggiungere la tolleranza di 10^{-6} posta sul residuo del PCG eseguite su CPU "Intel Xeon E5620" e GPU 'TESLA S2050"

si arresta quando il residuo relativo $res = \dfrac{\|Ax_m - b\|}{\|b\|}$ $m = 1, ..., n$ diventa minore della tolleranza prestabilita o quando viene raggiunto il numero di iterazioni massimo consentito posto uguale alla dimensione del problema n. Il numero di iterazioni per raggiungere la tolleranza prestabilita, nel caso sequenziale e parallelo, sono riportati nella tabella (3.1). Se osserviamo il caso in cui il sistema ha dimensione $n = 100000$, notiamo come il numero di iterazione utile per pervenire a tolleranza scala dallo 0.6% al 23% di n passando dal caso in cui $\alpha << \beta$ al caso $\alpha >> \beta$ come rappresentato in figura 3.7. Lo scopo dell'utilizzo del parallelismo delle GPU è quello di accelerare la risoluzione del sistema lineare (3.11) sia quando le dimensioni delle griglie crescono sia quando il rapporto delle funzioni α e β varia sul dominio. Nella tabella (3.2) sono riportati i tempi di risoluzione del sistema lineare al variare dei parametri suddetti. Si osserva come nel caso $n = 100000$ la versione parallela riduce il tempo di esecuzione di circa 6 volte rispetto alla versione sequenziale. E' possibile che questo fattore cresca con l'aumentare ulteriormente delle dimensioni del sistema (3.11).

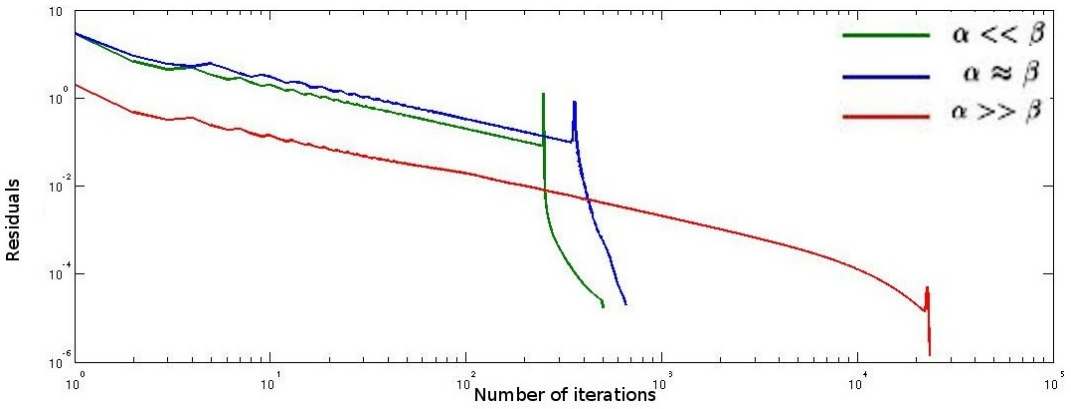

Figura 3.7: La figura riproduce l'andamento del residuo al variare del numero di iterazioni del PCG con precondizionatore diagonale per il caso $n = 100000$ quando $\alpha << \beta$, $\alpha \approx \beta$ e $\alpha >> \beta$

La tabella (3.2) riporta le accuratezze relative in norma 2 sulle soluzioni, in relazione al numero di iterazioni riportati in tabella (3.1), ottenute dal PCG, nel caso dell' implementazione seriale (CPU) e nel caso dell' implementazione parallela (GPU), al variare della dimensione n e del rapporto tra le funzioni α e β. Si osserva dalla tabella (3.2) come il numero di iterazioni incide anche significativamente sull'accuratezza della soluzione come mostrato nel caso più rappresentativo in cui la funzione α è molto maggiore della funzione β. Inoltre in tabella (3.3) sono mostrate le accuratezza delle soluzioni in norma infinito che forniscono informazioni sul massimo errore nelle componenti del vettore delle soluzioni.

Questi risultati indicano che quando le dimensioni dei sistemi lineari sono di elevate dimensioni e con elevata sparsità è opportuno utilizzare dei solutori iterativi implementati su GPU perché i risultati rivelano dei benefici da un punto di vista del tempo di esecuzione. Ci chiediamo ora se è pos-

n	**Version**	$\mathbf{time}_{\alpha<<\beta}$	$\mathbf{time}_{\alpha\approx\beta}$	$\mathbf{time}_{\alpha\approx\beta}$
1000	Seriale CPU	0.00	0.00	0.05
	Parallela GPU	0.09	0.13	0.88
10000	Seriale CPU	0.89	0.71	9.24
	Parallela GPU	0.85	0.68	8, 85
100000	Seriale CPU	7.71	10.89	1728, 67
	Parallela GPU	1.45	1.91	288.13

Tabella 3.2: La tabella mostra i tempi di esecuzione espressi in secondi del PCG per raggiungerela tolleranza prefissata nel numero di iterazioni riportati in tabella (3.1) nel caso sequenziale e nel caso parallelo rispettivamente su CPU "Intel Xeon E5620" e GPU "TESLA S2050"

n	**Versione**	$\mathbf{err-2}_{\alpha<<\beta}$	$\mathbf{err-2}_{\alpha\approx\beta}$	$\mathbf{err-2}_{\alpha>>\beta}$
1000	Sequenziale CPU	0.11×10^{-4}	0.53×10^{-6}	0.30×10^{-2}
	Parallela GPU	0.84×10^{-5}	0.68×10^{-6}	0.27×10^{-2}
10000	Sequenziale CPU	0.78×10^{-3}	0.79×10^{-5}	0.27×10^{0}
	Parallela GPU	0.78×10^{-3}	0.19×10^{-5}	0.27×10^{0}
100000	Sequenziale CPU	0.76×10^{-3}	0.38×10^{-5}	0.28×10^{1}
	Parallela GPU	0.76×10^{-3}	0.11×10^{-5}	0.25×10^{3}

Tabella 3.3: La tabella mostra le accuratezza in norma euclidea delle soluzioni ottenute nel caso seriale (CPU) e parallelo (CUDA) del PCG al variare della dimensione n e del rapporto tra le funzioni α e β legate alla precedente tabella (3.1) su CPU "Intel Xeon E5620" e GPU 'TESLA S2050"

sibile accelerare ancora i nuclei risolutivi incrementando anche la rapidità di convergenza di questi, cambiando opportunamente su queste architetture, il precondizionatore. I precondizionatori di tipo AINV appartenenti alla libreria CUSP, basati sulle tecniche di precondizionamento di Bridson [19], rappresentano una buona soluzione per questo problema. Negli esperimenti numerici condotti si è posto con P il precondizionatore diagonale, con P' il precondizionatore di Bridson, ottenuto imponendo una *drop tolerance* di 10^{-1}, con P'' il precondizionatore di Bridson ottenuto imponendo una *stat-*

n	**Versione**	**err** $-\infty_{\alpha<<\beta}$	**err** $-\infty_{\alpha\approx\beta}$	**err** $-\infty_{\alpha>>\beta}$
1000	Sequenziale CPU	0.17×10^{-4}	0.14×10^{-5}	0.42×10^{-2}
	Parallela GPU	0.17×10^{-4}	0.21×10^{-5}	0.36×10^{-2}
10000	Sequenziale CPU	0.11×10^{-2}	0.13×10^{-4}	0.30×10^{0}
	Parallela GPU	0.11×10^{-2}	0.38×10^{-5}	0.30×10^{0}
100000	Sequenziale CPU	0.11×10^{-2}	0.11×10^{-4}	0.97×10^{0}
	Parallela GPU	0.19×10^{-2}	0.21×10^{-4}	0.25×10^{3}

Tabella 3.4: La tabella mostra le accuratezza delle soluzioni ottenute in norma infinito nel caso sequeziale (CPU) e parallelo (CUDA) del PCG al variare della dimensione n e del rapporto tra le funzioni α e β legate alla precedente tabella (3.1) su CPU "Intel Xeon E5620" e GPU 'TESLA S2050"

ic dropping tolerance, che consiste nel prestabilire il numero di elementi da preservare su ogni riga di P'', che nel nostro caso si è convenuto porre a 10 e infine si è posto con P''' il precondizionatore di Bridson dotato di una novella *dropping strategy* come in [73] ed infine il precondizionatore $\widehat{P}'$ costruito nel paragrafo (2.5) del capitolo (2). Gli esperimenti numerici, eseguiti utilizzando ancora una volta la singola precisione sono stati condotti sempre sulla scheda grafica a cui si è fatto riferimento all'inizio della sezione ovvero la "TESLA S2050". Fissata una tolleranza sul residuo relativo di 10^{-6}, il numero di iterazioni del PCG con i precondizionatori su detti, al variare della dimensione del problema e delle funzioni α e β, è riportato nella tabella (3.5). Si osserva nella tabella, nel caso particolare di $n = 100000$, come il precondizionatore P' sia il più efficiente nel caso in cui $\alpha << \beta$ e $\alpha >> \beta$, ottenendo una diminuzione del numero di iterazioni del $96,8\%$ nel primo caso e del $98,3\%$ nel secondo rispetto al precondizionatore diagonale. Sempre dalla tabella (3.5) si osserva come il precondizionatore P'' sia il più efficienti nel caso in cui $\alpha \approx \beta$, riducendo il numero di iterazioni del 70% rispetto

al precondizionatore P. Questa riduzione del numero di iterazioni ha effetti anche sul tempo di risoluzione come mostrato in tabella (3.6). Se si fa ancora riferimento al caso $n = 100000$, con il precondizionatore P' si ottiene una riduzione del tempo nel caso $\alpha << \beta$ e $\alpha >> \beta$ rispettivamente del $84,33\%$ e del $87,53\%$ rispetto al precondizionatore diagonale. Con il precondizionatore P'' per $\alpha \approx \beta$ si ottiene invece una riduzione del 47% rispetto al precondizionatore diagonale. Osserviamo in fine che le riduzioni prestazionali dal punto di vista del tempo non coincidono con la riduzione dal punto di vista delle iterazioni in quanto il costo di implementazione di P' e soprattutto di P'' è maggiore di quello di P nel PCG. I risultati indicano come la scelta del precondizionatore sia fortemente legata anche alle variazioni del rapporto delle funzioni α e β come mostrato in figura (3.8).

n	**Precond.**	**iter**$_{\alpha<<\beta}$	**iter**$_{\alpha\approx\beta}$	**iter**$_{\alpha>>\beta}$
1000	$P_{diagonal}$	56	83	510
	$\widehat{P}'_{inverse}$	56	58	291
	$P'_{inverse}$	8	47	21
	$P''_{inverse}$	13	31	88
	$P'''_{inverse}$	20	38	167
10000	$P_{diagonal}$	501	402	4999
	$\widehat{P}'_{inverse}$	501	289	2549
	$P'_{inverse}$	16	214	144
	$P''_{inverse}$	75	144	659
	$P'''_{inverse}$	103	216	904
100000	$P_{diagonal}$	499	471	11901
	$\widehat{P}'_{inverse}$	499	654	23334
	$P'_{inverse}$	16	342	403
	$P''_{inverse}$	64	192	2695
	$P'''_{inverse}$	87	312	3794

Tabella 3.5: La tabella riporta il numero di Iterazioni del PCG con precondizionatore diagonale e con precondizionatori inversi al variare della dimensione n di A e del rapporto delle funzioni α e β, fissata una tolleranza relativa sul residuo di 10^{-6}

n	**Precond.**	$\mathbf{time}_{\alpha\leq\beta}$	$\mathbf{time}_{\alpha=\beta}$	$\mathbf{time}_{\alpha\geq\beta}$
1000	$P_{diagonal}$	0.07×10^0	0.09×10^0	0.62×10^0
	$\widehat{P}'_{inverse}$	0.07×10^0	0.063×10^0	0.35×10^0
	$P^{'}{}_{inverse}$	0.01×10^0	0.09×10^0	0.05×10^0
	$P^{''}{}_{inverse}$	0.03×10^0	0.07×10^0	0.18×10^0
	$P^{'''}{}_{inverse}$	0.04×10^0	0.08×10^0	0.34×10^0
10000	$P_{diagonal}$	0.62×10^0	0.50×10^0	6.18×10^0
	$\widehat{P}'_{inverse}$	0.62×10^0	0.36×10^0	3.15×10^0
	$P^{'}{}_{inverse}$	0.05×10^0	0.28×10^0	0.46×10^0
	$P^{''}{}_{inverse}$	0.12×10^0	0.10×10^0	0.97×10^0
	$P^{'''}{}_{inverse}$	0.16×10^0	0.14×10^0	1.28×10^0
100000	$P_{diagonal}$	1.34×10^0	1.73×10^0	61.42×10^0
	$\widehat{P}'_{inverse}$	1.34×10^0	1.24×10^0	31.33×10^0
	$P^{'}{}_{inverse}$	0.21×10^0	1.11×10^0	7.66×10^0
	$P^{''}{}_{inverse}$	0.28×10^0	0.90×10^0	12.43×10^0
	$P^{'''}{}_{inverse}$	0.36×10^0	1.41×10^0	16.48×10^0

Tabella 3.6: La tabella mostra il tempo di esecuzione in secondi del PCG con precondizionatore diagonale e con precondizionatori inversi relativo alle iterazioni in tabella (3.6) su architettura grafica “TESLA S2050”

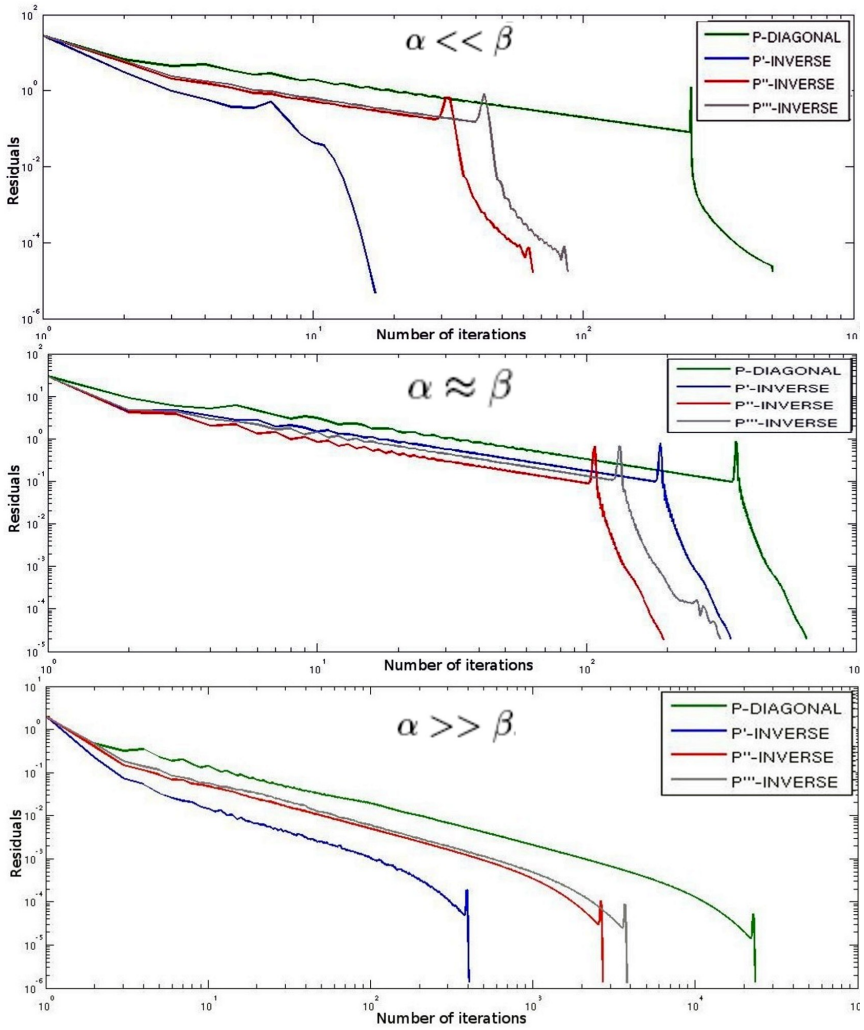

Figura 3.8: La figura riproduce l'andamento del residuo del PCG con precondizionatore diagonale e precondizionatori inversi per il caso $n = 100000$ quando $\alpha << \beta$, $\alpha \approx \beta$ e $\alpha >> \beta$

3.6 Conclusioni

In questo capitolo abbiamo provato come l'uso del GPU computing può migliorare la performance dei solutori iterativi nei problemi numerici dei modelli oceanici globali. Nel particolare in questo capitolo è stato analizzato un sistema lineare $Ax = b$ ottenuto dal nucleo ellittico del modello oceanico OPA-NEMO per mezzo delle differenze finite. Il sistema lineare è risolto in principio mediante un' implementazione del PCG con precondizionatore diagonale sulle GPU dando ottimi risultati da un punto di vista prestazionale. Inseguito a causa di una lenta convergenza del solutore iterativo, quando le dimensioni delle griglie di risoluzioni crescono o quando il dominio del modello contiene zone del pianeta in prossimità dei poli terrestri dove c'è un forte sbilanciamento tra le funzioni α e β, la tecnica di precondizionamento è stata sostituita mediante precondizionatori di tipo inversi ottenendo ottimi risultati sia da un punto di vista della convergenza che del tempo di esecuzione del solver. I risultati ottenuti mostrano alla comunità oceanografica come una efficiente tecnica di precondizionamento combinata con il GPU computing possono accelerare i solutori iterativi di questi modelli oceanici, dando la possibilità di andare oltre nelle simulazioni.

Appendice A

Appendice

A.1 L'Architettura NVIDIA Tesla

In questo paragrafo ci occuperemo del GPU computing di NVIDIA, analizzando il suo modello di programmazione CUDA (CUDA Basic Linear Algebra Subprograms) e l'architettura hardware sulla quale si basa. In figura (A.1) è schematizzata l'architettura Tesla[1], la quale sta alla base anche della GPU NVDIA *S*2050 utilizzata per i nostri esperimenti nella sezione (2.6). Le componenti principali di una architetture Tesla consistono in una memoria Dynamic Random Access Memory (DRAM) e di uno Scalable Processor Array (SPA) (figura A.1). Lo SPA è la componente che si occupa di eseguire tutte le operazioni programmabili sulla GPU. Nello schema (figura A.1) la computazione va dall'alto verso il basso, attraverso tutte le componenti, alcune delle quali però sono dedicate al rendering e rimangono quindi inutilizzate per le operazioni di computing. In questa sezione ci soffermeremo sulle componenti della GPU che hanno un ruolo determinante per fini

[1]ll termine Tesla viene utilizzato da NVIDIA per indicare una particolare tecnologia relativa alle proprie GPU che ha lo scopo di utilizzare la potenza elaborativa di tali soluzioni in ambiti relativi al calcolo ad alte prestazioni.

computazionali.

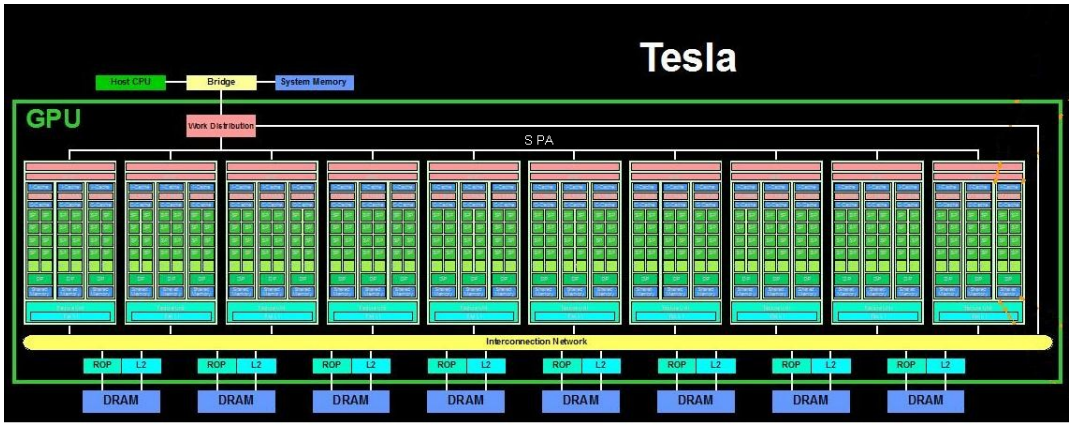

Figura A.1: Modello di GPU basato sull'architettura NVIDIA TESLA

A.2 Schema delle Operazioni su GPU

Utilizzando una raffigurazione più dettagliata della architettura Tesla in figura (A.2), l'Host Interface è la componente dell'architettura si occupa della comunicazione tra Host e la Device, ovvero tra CPU e la GPU. Il compito principali del Host Interface è quello di trasferire i dati tra la memoria della CPU e quella della Device e viceversa. Successivamente il Compute Work Distribution (figura A.2) si occupa della distribuzione sullo SPA del flusso di istruzioni generato dall' esecuzione dei kernel. I kernel sono quelle parti di programmi parallelizzabili mediante le GPU. Alla fine dell' elaborazione su SPA, i risultati dei calcoli passano ai Texture Processor Cluster (ROP) attraverso un network di connessione. I ROP hanno il compito di eseguire le funzioni non programmabili di colorazione dei pixel all'interno delle schede grafiche, ed operano direttamente sulla memoria DRAM alla quale sono connessi.

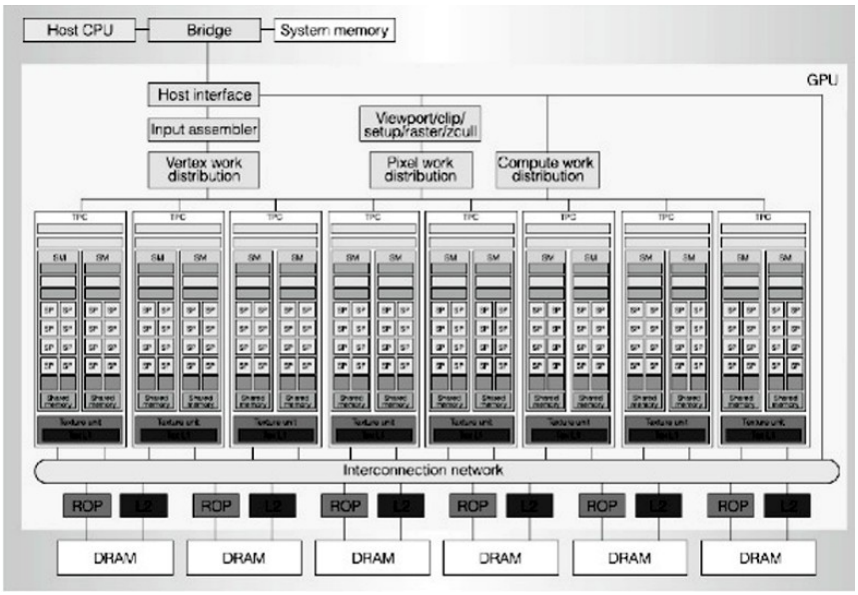

Figura A.2: Schema della Tesla

A.3 Le Componenti dello SPA

Lo Scalable Processor Array (SPA) è costituito dalle Texture Processor Cluster (TPC), il cui schema è rappresentato a sinistra di Figura (A.3), il numero di queste componenti dipende dalle caratteristiche specifiche e dalla classe di prestazioni di ciascuna GPU considerata. All'interno della TPC vi sono differenti componenti, qui di seguito elencheremo le principali.

- Stream Multiprocessor (SM). Lo SM rappresentato (figura a destra di (A.3)), è un processore in grado di eseguire istruzioni per lo GPU computing. Ogni SM contiene 8 SP cores (indipendente dalla fascia di mercato del chip grafico considerato), 2 Special Function Unit (SFU), una Multi Threaded instruction fetch e Issue unit (MT) issue, una cache delle istruzioni, una cache per la memoria costante, 16 KByte

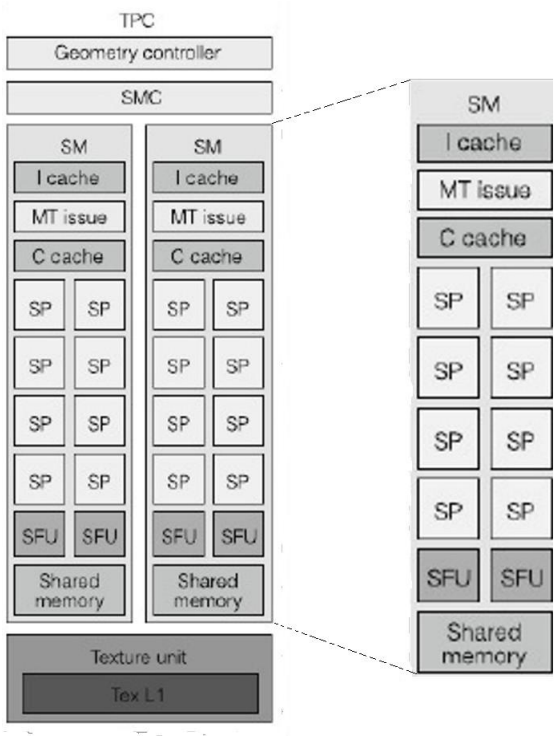

Figura A.3: A sinistra schema di composizione della Texture Processor Cluster (TPC) mentre a destra quello dello Stream Multiprocessor (SM)

di memoria condivisa ed 8192 registri da 32 bit ciascuno. Ogni SP contiene un'unità scalare Multiply-Add (MAD), dando quindi 8 unità MAD ad ogni SM. Le due SFU nella SM sono usate per il calcolo delle funzioni trascendentali (esponenziali, logaritmiche, trigonometriche) ed inoltre ogni SFU contiene 4 unità per la moltiplicazione per numeri in virgola mobile. Le operazioni di load e store dalla memoria principale degli SP sono implementante dal Streaming Multiprocessor Controller (SMC) del TPC, mentre le operazioni di accesso alla memoria condivisa (Shared Memory), situato sullo SM, sono effettuate direttamente.

- Texture. L'unità delle texture è una componente che permette di accedere in sola lettura alla memoria DRAM. Questa è dotata di un meccanismo di caching[2] tale da privilegiare gli accessi localizzati ai dati. All' interno di un TPC, la Texture unit (figura A.3) serve contemporaneamente le due SM, fornendo un' alternativa all'uso dello SMC per la lettura dei dati.

- Shared Memory. La mancanza di un sistema di caching generale, della memoria DRAM(figura (A.4)), favorisce l' ampiezza di banda passante dei dati tra GPU e memoria. Allo stesso tempo però la mancanza di una cache comporta un' alta latenza per le operazioni sui dati in memoria, superiore di $400 - 600$ volte al tempo di accesso ai registri locali. Tuttavia grazie ad uno scheduling intelligente delle operazioni, la latenza di accesso alla DRAM viene nascosta quando si ha un elevato numero di *thread*[3] per gli SM. I tempi di accesso alla memoria condivisa (shared memory figura (A.4)) , al contrario della memoria globale, sono di poco superiori a quelli dei registri. La bassa latenza è dovuta al fatto che i 16 Kbyte di memoria condivisa sono situati all' interno di ogni SM e non esternamente. Tuttavia questi 16 Kbyte di memoria condivisa sono partizionati in 16 banchi da 1 Kbyte, ciascuno soggetto a conflitti di accesso da parte degli SP cores componenti dello SM. Infatti nel

[2]Un meccanismo di caching è un meccanismo che si appoggia generalmente ad una memoria piccola e veloce (cache) per ridurre i tempi di accesso alla memoria globale

[3]Un *thread* è un flusso di esecuzione all' interno di un'applicazione. Quando il flusso è unico si parla di applicazione single *thread*, altrimenti, quando i flussi sono più di uno, si parla di applicazioni *multi thread*

caso ci siano più SP cores ad accedere allo stesso banco, le operazioni di ciascuno di questi vengono serializzate, con un conseguente degrado delle prestazioni dovuto all' aumento della latenza.

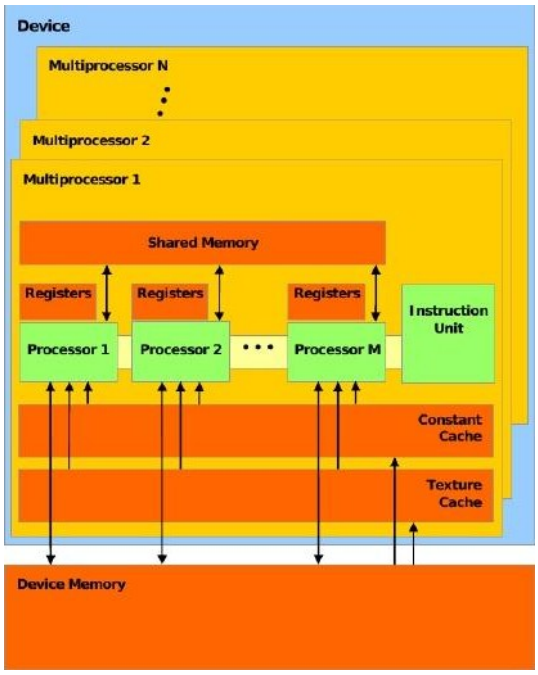

Figura A.4: Schema degli accessi in Memoria sulla Device

A.4 Un' architettura di tipo SIMT

Dalla punto di vista implementativo NVIDIA usa delle unità Single Instruction Multiple Data (SIMD) composte da 8 cores ciascuno, però al livello funzionale queste non operano eseguendo 8 volte la stessa istruzione per un solo flusso di esecuzione, come farebbero delle unità SIMD tradizionali. Esse, infatti, gestiscono 8 flussi di esecuzione differenti, eseguendo un'istruzione per ciascuno di essi. I singoli SP cores quindi si comportano come delle unità

scalari e non come unità di esecuzione di un processore vettoriale. Per evidenziare questa differenza, NVIDIA parla di un'architettura SIMT. A differenza delle unità SIMD, questa è in grado di massimizzare l'uso delle proprie unità di calcolo, soprattutto quando l'applicazione è composta da molti *thread*. La GPU infatti viene utilizzata a pieno solo quando i *thread* sono complessivamente nell' ordine delle migliaia, grazie al fatto che questi hanno un costo di creazione e gestione praticamente nullo. L'architettura lavora assegnando ad ogni flusso di istruzioni un SP core, in modo che ogni *thread* venga eseguito indipendentemente dagli altri, conservando il proprio stato dei registri e indirizzo delle istruzioni. Ogni SM gestisce ed esegue simultaneamente fino 768 thread divisi in 24 gruppi gruppi da 32 thread, chiamati Warps, per i quali la MT Issue si occupa di selezionare e inoltrare le istruzioni. Durante l'esecuzione di un programma è possibile che, in presenza di un'istruzione condizionale (if-else,switch-case, ecc), due o più *thread*, all'interno dello stesso Warp, seguano percorsi differenti. Quando tutti i 32 *thread* di un Warp seguono lo stesso ramo di esecuzione, questa architettura raggiunge il massimo dell' efficienza per il fatto che tutti gli SP cores di un SM eseguono la medesima istruzione. Invece, se i *thread* all' interno di un Warp divergono, le prestazioni decadono. Infatti, quando si presenta quest' ultimo caso, la MT Issue esegue in serie le istruzioni per i diversi flussi, fintanto che questi rimangono separati. Scrivere un software che tenga conto di questa caratteristica è fondamentale se si vuole ottenere il massimo delle prestazioni dalla GPU di NVIDIA. Nonostante la maggiore efficienza, NVIDIA non vede in un' architettura SIMT la soluzione definitiva per le proprie GPU, poichè es-

sa introduce diversi problemi rispetto ad architetture più semplici. Facendo un confronto con la classe dei sistemi SIMD è infatti richiesta una maggiore logica di controllo, che comporta un aumento della superficie del chip e un maggior consumo di energia.‘ E sufficiente osservare i dettagli tecnici delle architetture, prima di AMD, e poi di NVIDIA, per vedere l' enorme differenza nella superficie dei chip che, nel primo caso, si limita a 192 mm^2 contro i 484 mm^2 del secondo.

A.5 Aritmetica Floating Point nelle GPU

L' architettura Tesla contiene un vasto insieme di istruzioni per operazioni su numeri interi, a virgola mobile, singoli bit, conversioni di tipo, funzioni trascendentali, controllo di flusso , lettura-scrittura dalla memoria e operazioni sulle texture. Di maggiore interesse per il calcolo scientifico sono le istruzioni che operano sui numeri in virgola mobile, che seguono lo standard di rappresentazione *IEEE* 754. Nell'elenco seguente si mostrano le differenze principali tra l'implementazione interna di Tesla e le specifiche ufficiali dello standard *IEEE* 754. La lista completa di queste differenze si può consultare in [81] qui noi ne presentiamo alcune.

- Moltiplicazione e addizione possono essere combinate in una sola istruzione FMAD che non tiene conto del risultato intermedio della moltiplicazione.

- La divisione è implementata attraverso la moltiplicazione per il reciproco.

- L' operazione di radice quadrata di un numero è implementata come il reciproco della radice quadrata dell'inverso del numero stesso.

- La politica di recovery adottata per le condizioni di errore è la store 0[4].

Queste ed altre piccole differenze possono portare ad una discordanza con i risultati dei calcoli eseguiti da una CPU. Tuttavia gli errori di moltiplicazione e addizione non superano mai $\frac{1}{2}u$ con u precisione della macchina ed errori maggiori si possono verificare solo nel calcolo di funzioni trascendenti, i quali però restano sempre limitate da $3 * u$.

A.6 Il Modello di Programmazione CUDA

CUDA è un modello di programmazione parallela ed un ambiente software pro- gettato per sfruttare le capacità dei nuovi sistemi paralleli multicore e, più in particolare, delle GPU ad architettura Tesla. I concetti principali sui quali questa piattaforma si basa sono:

- Gerarchia dei Thread,

- Memoria condivisa,

- Sincronizzazione a barriera.

Questi vengono controllati dal programmatore mediante poche estensioni al linguaggio C, facilitando cosi l'apprendimento per chi ha già ha familiarià con questo ambiente. Le estensioni indirizzano lo sviluppatore a partizionare

[4]Lo store 0 prevede nel caso che un' operazione incorra in un errore di underflow, questa tecnica ne azzera il risultato. Al contrario del gradual underflow, adottato nella maggior parte delle CPU per personal computer.

un problema in più sotto problemi risolvibili in maniera indipendente. Tali problemi, a loro volta, vengono risolti mediante la cooperazione di più *thread*, ovvero, di più flussi d'esecuzione concorrenti dello stesso programma. Questa soluzione permette sia la cooperazione fra gruppi di *thread* per la risoluzione di ogni sottoproblema, sia l' assegnazione di un sottoproblema indipendente ad uno qualunque dei processori disponibili. La piattaforma è quindi scalabile rispetto, sia al numero di SM presenti nel sistema, sia al numero di *thread* che questi possono gestire.

A.7 Gerarchia dei Thread

CUDA estende quindi il linguaggio *C* permettendo al programmatore di definire delle funzioni speciali chiamate kernel che, quando invocate, verranno eseguite concorrentemente in N blocchi differenti ciascuno composto da più *thread*, come rappresentato in figura (A.5).

Ogni *thread* possiede una variabile *threadIdx*, accessibile all' interno del kernel ed inizializzata univocamente rispetto alla stessa variabile degli M *thread* dello stesso blocco. La variabile *threadIdx* è una variabile avente 3 componenti intere, cosi che i *thread* possano essere organizzati in spazi monodimensionali, bidimensionali o tridimensionali. Questo permette al programmatore di scegliere la struttura di esecuzione del kernel più conveniente, in base al problema da risolvere. Il codice in esempio 10 mostra come si effettua la somma tra due due vettori di lunghezza M, A e B, ponendo il risultato in un terzo vettore C. Ogni *thread* somma una componente di A con la rispettiva componente di B e ne scrive il risultato in C. Essendo i *thread* organizzati in

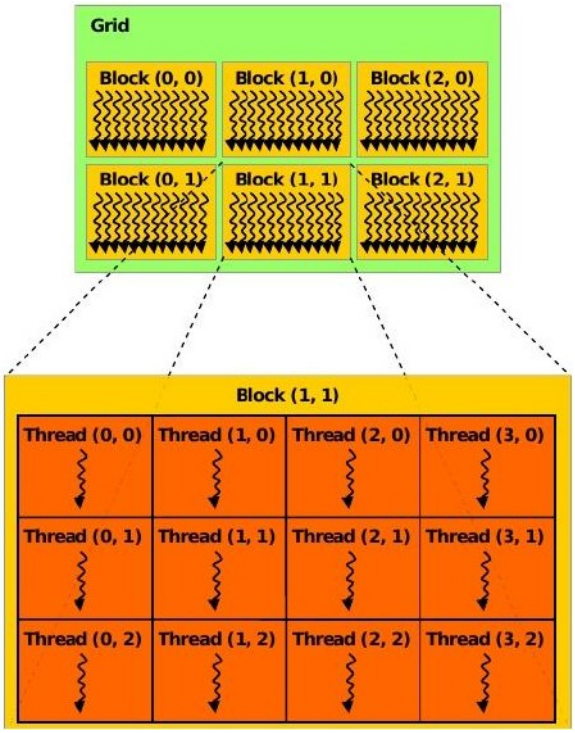

Figura A.5: Gerarchia dei *thread*

uno spazio monodimensionale, la componente dei vettori sulla quale opera ognuno di questi è data dal primo dei tre valori di *threadIdx*. Durante l' esecuzione, per ogni *thread*, questo valore sarà distinto e compreso tra 0 e $M - 1$.

Nell'esempio (10) sono dichiarate due funzioni. All'interno della funzione main, si invoca la funzione vecAdd, la quale riceve in input 3 riferimenti a vettori. Lo specificatore __global__ e ed i parametri contenuti tra i caratteri <<< e >>>, indicano rispettivamente al compilatore che la funzione vecAdd verrà eseguita sulla GPU in un solo blocco composto da M *thread* concorrenti. Analogamente a quanto detto per *threadIdx*, si ha che ai *thread* appartenenti allo stesso blocco è assegnata la variabile *blockIdx*, accessibile al-

Schema 10 Programma vecAdd

```
1   __global__void__vecAdd(* float A, * float B, *float C){
2 {
3  int i= threadIdx.x;    //prima componente della variabile threadIdx
4
5  C[i] = A[i]+B[i];
6
7 }
8 }
9  host int main(){
10
11  vecAdd<<<1, M>>>(A, B, C);
12
13 }
```

l'interno dei kernel e contenente il numero di blocco a cui appartiene il *thread* stesso. Questa variabile è composta da una o due componenti, a seconda che i blocchi siano organizzati logicamente in uno spazio monodimensionale o bidimensionale. Il codice (10) estende il codice (11) , introducendo l'uso di blocchi organizzati in uno spazio monodimensionale

Schema 11 Programma vecAdd2

```
1  __global__void__vecAdd(* float A, * float B, *float C){
2 {
3
4 int i= blockDim.x*blockIdx.x+threadIdx.x; //calcolo dell' id assoluto
5
6  C[i] = A[i]+B[i];
7 }
8
9 }
10  host int main(){
11
12  vecAdd<<<N, M>>>(A, B, C);
13
14 }
```

In questo esempio la dimensione dei vettori A e B è di $M \times N$ elementi, ed i parametri contenuti tra i caratteri <<< e >>> indicano al compilatore

che verranno eseguiti N blocchi composti da M *thread* ciascuno. Al fine di assegnare a tutti i *thread* la somma di una componente univoca dei due vettori, l' assegnazione di tale componente non dipenderá più solo dalla variabile *threadIdx*, ma dipenderà anche dalla variabile *blockIdx*. In questo modo gli M *thread* appartenenti agli N blocchi differenti, sono in grado di sommare distintamente tutti gli elementi dei vettori A e B.

A.8 Configurazione di Esecuzione del Kernel

La configurazione di esecuzione di un kernel dipende dal numero dei blocchi che verranno eseguiti sulla GPU, ed dal numero dei *thread* per ogni blocco. All' interno di un kernel, è possibile stabilire tale configurazione mediante le variabili *blockDim* e *gridDim*: la prima contenente il numero di *thread* per blocco e la seconda contenente il numero di blocchi per il kernel in esecuzione. L' architettura Tesla delle GPU, come altre, limita il numero di *thread* per blocco a 512, e prevede al massimo la gestione 65536 blocchi contemporaneamente. Lo scheduling dei blocchi di un kernel è gestito in hardware, sulla base di un algoritmo che di volta in volta assegna ogni blocco ad un solo SM, e ad ogni SM assegna contemporaneamente al più 8 blocchi, la cui dimensione complessiva dei thread non deve superare però i 768 *thread* (24 warps). Questi blocchi vengono eseguiti assieme fino a che lo scheduler decide di sospenderne l'esecuzione e assegnare allo SM dei nuovi blocchi. Per calcolare l' efficienza di una configurazione si introduce il concetto di occupazione, il quale è facilmente esprimibile come il rapporto tra il numero di warps attivi

e il numero massimo di warps gestibili da un SM,

$$occupacy = \frac{active\ warps}{maxactive\ warps}. \tag{A.1}$$

Configurazioni che offrono l' occupazione massima possono essere facilmente costruite. Le più efficienti per le GPU basate sulle architettura Tesla sono quelle che hanno almeno 96 *thread* per blocco. In questo modo lo scheduler è in grado di mascherare buona parte della latenza di lettura e scrittura, sia della memoria, sia dei registri. Configurazioni con molti *thread* per blocco, però, introducono delle limitazioni sul numero dei registri utilizzabili all' interno di un kernel, ad esempio: se si decide per una configurazione di 256 *thread* per blocco, scrivere un kernel che usa 10 registri invece di 11 permette di passare da 2 a 3 blocchi (da 16 a 24 warps) attivi contemporaneamente per ogni singolo SM. Infatti, considerando 3 blocchi in esecuzione per ogni SM, nel caso ogni *thread* usi 10 registri, i registri utilizzati in totale sono $3 \times 256 \times 10 = 7680$; un valore entro il limite degli 8192 disponibili. Invece, considerando 3 blocchi per ogni SM, ma assumendo che ogni *thread* usi 11 registri, questa volta i registri utilizzati in totale sono $3 \times 256 \times 11 = 8448$, ovvero troppi affinchè 3 blocchi possano essere attivi contemporaneamente su un solo SM.

A.9 Gerarchia della Memoria

Ogni *thread* ha accesso, in lettura e scrittura, a più spazi di memoria della SM durante la sua esecuzione:

- **Registri** Su ogni SM sono allocati 8192 registri e, ad ogni *thread*, ne è concesso solo un numero limitato. Le operazioni sui registri hanno latenza praticamente nulla, anche se possono avere dei ritardi di lettura dopo una scrittura dovuti all' aggiornamento del loro contenuto. Il numero di registri disponibili per ogni *thread* varia in base alla configurazione di esecuzione che si sceglie. Fissata quest'ultima, il loro numero può essere calcolato mediante la formula seguente:

$$Regth = \frac{Reg}{BlockAt \times ceil(BlockDim, 32)} \tag{A.2}$$

  dove $Regth$ è il numero di registri per *thread*, Reg è il numero di registri per ogni SM , $BlockAt$ è il numero di blocchi attivi per SM e $BlockDim$ è il numero di *thread* per blocco, che, come si vede, viene approssimato per eccesso con il più piccolo multiplo di 32.

- **Memoria Globale** Questa è la memoria che i *thread* utilizzano, sia per leggere i dati da elaborare, sia per scrivere i risultati delle loro operazioni. Essa è implementata nella DRAM, quindi i suoi tempi di accesso, come già visto durante lo studio dell'architettura Tesla, sono centinaia di volte superiori a quelli dei registri: è importante ridurne l'utilizzo il più possibile. Quando l'accesso alla memoria globale non è evitabile, è conveniente usare delle tecniche per leggere o scrivere i dati in blocchi detti accessi *coalesced*. Nelle GPU della serie basata su Tesla, un' operazione di lettura o scrittura *coalesced* si ha quando almeno la metà dei *thread* di un warp (half-warp) effettua un accesso alla memoria, in modo tale che il *thread* $i-$esimo acceda alla locazione $i-$esima

di uno spazio allineato ad un indirizzo multiplo delle dimensioni di un half-warp, ovvero 16. Operazioni di lettura e scrittura *coalesced* hanno tempi inferiori di circa 10 volte rispetto ad operazioni di lettura e scrittura non strutturate.

- **Memoria Locale Privata** - In questa memoria vengono allocate le variabili locali dei singoli *thread*, che non possono essere contenute nei registri. Anche questo è uno spazio di memoria implementato nella DRAM e, di conseguenza, i tempi di accesso sono gli stessi della memoria globale.

- **Memoria Condivisa** - Questa memoria è accessibile a tutti i *thread* dello stesso blocco e, generalmente, è utilizzata per condividere i risultati intermedi dei processi di calcolo. Questa memoria è implementata mediante i banchi di memoria condivisa contenuti all' interno di ogni SM, quindi la latenza è leggermente maggiore di quella dei registri. Spazi contigui di memoria condivisa sono allocati su banchi differenti in modo da ridurre i conflitti di accesso e, di conseguenza, anche i tempi di latenza.

I seguenti sono spazi di memoria accessibili in sola lettura

- Memoria Costante - Questa memoria è uno spazio di memoria implementato nella memoria globale, e contiene quei valori che restano costanti durante tutta l'esecuzione del kernel. I tempi di latenza sono al pari di quanto visto per la memoria globale; tuttavia questo spazio di

memoria dispone di una cache. La sua presenza riduce drasticamente i tempi di attesa nel caso si acceda molte volte allo stesso elemento.

- **Memoria delle Texture** - Questo spazio di memoria è accessibile attraverso l' unità delle texture. Rappresenta quindi un'interfaccia di sola lettura alla memoria globale, che offre un meccanismo di caching ottimizzato per la località bidimensionale e, facoltativamente, un diverso tipo di indirizzamento ai dati [5]. Nonostante le latenze siano superiori agli accessi di tipo *coalesced* verso la memoria globale, l'uso della memoria delle texture può essere molto vantaggioso. Tramite questo spazio di memoria infatti, si possono eseguire operazioni di lettura che, o non possono essere rese *coalesced*, o coinvolgono un insieme di dati sufficientemente piccolo da essere contenuto interamente nella cache.

Gli spazi di memoria globale, costante e delle texture sono consistenti tra la chiamata di un kernel e un altro. Al contrario, i contenuti della memoria locale e della memoria condivisa sono eliminati al termine di ogni esecuzione.

A.10 Sincronizzazione

I *thread* all' interno dello stesso blocco possono cooperare tra di loro, condividendo dati attraverso la memoria condivisa, e sfruttando la primitiva di sincronizzazione a barriera *__syncthreads()__*. Quest'ultima, dopo essere stata chiamata all'interno di un *thread*, ne blocca l'esecuzione, fino a che tutti gli altri *thread* appartenenti allo stesso blocco non l' hanno invocata a loro vol-

[5] Ulteriori informazioni sulla memoria delle texture si trovano in [81], alla quale è stata dedicata un'intera appendice

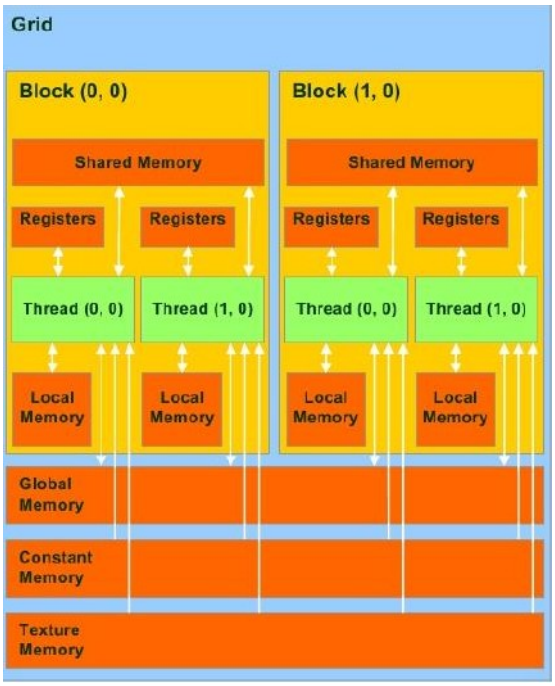

Figura A.6: Gerarchia della memoria in CUDA

ta. L' uso di *__syncthreads()__* assume un ruolo determinante quando, ad un dato momento del programma, si vuol essere sicuri che tutti i processi di un blocco abbiano terminato il loro compito; ad esempio la scrittura dei propri dati in memoria. Questa funzione è l'unico meccanismo di sincronizzazione offerto da CUDA. La mancanza di una primitiva di sincronizzazione globale costringe ad attendere il ritorno della chiamata ad un kernel, per avere la certezza che tutti i blocchi abbiano terminato il proprio compito.

A.11 Uso della Piattaforma

Prima di terminare questa sezione, è importante sottolineare come CUDA sia stato concepito per sfruttare le capacità di calcolo parallelo delle GPU come se queste fossero dei coprocessori della CPU principale. Pertanto l'Host

si occupa sia dei trasferimenti dei dati da e verso la DRAM del Device, sia della configurazione e dell' esecuzione dei kernels. Un esempio del flusso di un'applicazione CUDA è mostrato in dove si osserva l'alternanza tra il codice seriale eseguito sull' Host, e l'esecuzione parallela di uno o più kernel sulla Device.

A.12 Il Modello di Programmazione CUBLAS

Le CUBLAS [80] sono librerie che si poggiano sui driver NVIDIA CUDA e possono essere utilizzate senza un'interazione diretta con le istruzioni vere e proprie di CUDA. Lo scopo di queste librerie è quello di fornire al programmatore delle funzioni di algebra lineare che, venendo eseguite nella GPU, sfruttano l'iperparallelismo che la scheda grafica mette a disposizione per eseguire le operazioni con un grande risparmio di tempo. Il modello di programmazione CUBLAS si basa fondamentalmente su tre step principali:

1. creazione delle strutture dati in GPU (matrici o vettori). Per effettuare questa operazione è necessario prima di tutto allocare in GPU lo spazio necessario a contenere i dati attraverso l'istruzione *cublasAlloc()*, e poi sfruttare questo spazio per copiare i dati da CPU in una vettore o in una matrice attraverso le istruzioni *cublasSetVector()* e *cublasSetMatrix()*;

2. Modifica dei dati caricati sul device attraverso le funzioni per l'algebra lineare messe a disposizione dalle CUBLAS. Queste si dividono in BLAS (Basic Linear Algebra Subprograms) di livello 1,2,3 [16], a

loro volta suddivise in funzioni a singola precisione (operanti su float), doppia precisione (operanti su double) e funzioni su numeri complessi ;

3. Aggiornamento dei dati in CPU attraverso le primitive *cublasGetVector()* e *cublasGetMatrix()* e deallocazione dello spazio in GPU attraverso la primitiva *cublasFree()*.

Inoltre è necessario avviare le CUBLAS attraverso *cublasInit()* prima di utilizzare qualsiasi operazione *CUBLAS* e poi spegnerle alla fine con *cublasShutdown()*. CUBLAS inoltre è dotato di un sistema per recuperare e comprendere gli errori che avvengono in GPU durante l'esecuzione delle operazioni. Ogni funzione CUBLAS restituisce un tipo di dato *cublasStatus* che descrive come è avvenuta l'operazione, ottenibile anche attraverso la funzione *cublasGetError()*.

Bibliografia

[1] Fox L., H. D. Huskey, and J. H. Wilkinson,. Notes on the solution of algebraic linear simultaneous equations. *Quart. J. Mech. Appl. Math., 1* (1948).

[2] Adcroft A., Campin J.-M., Heimbach P., Hill C., Marshall J. MITgcm user's manual. *(online documentation). http://www.mitgcm.org.* (2007).

[3] Arakawa A., F.Mesinger. Numerical Methods used in Atmospheric Models. *VOL 1 Garp Publication Series NO 17, France (1976).*

[4] Asselin R. Frequency Filter for Time Integrations. *Mon. Weather Rev., 100,.* (1972).

[5] Axelsson O. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT Numerical Mathematics* (1985).

[6] Axelsson O. *Iterative Solution Methods.* Cambridge University Press, Cambridge, 1994.

[7] Barnard S. T. , L. M. Bernardo, and H. D. Simon,. An MPI implementation of the SPAI preconditioner on the T3E. *Int. J. High Perform. Comput. Appl. 13* (1999).

[8] Beare M.I. The Southampton - East Anglia (SEA) Model: A General Purpose Parallel Ocean Circulation Model, in: High Performance Computing. *Plenum Publishing Company Ltd., London*.

[9] Bell N. and M. Garland. Efficient sparse matrix-vector multiplication on CUDA. *NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation*.

[10] Bell N. and M. Garland. Efficient sparse matrix-vector multiplication on CUDA. *NVIDIA Technical Report NVR-2008- 004, NVIDIA Corporation* (2008).

[11] Benzi M. A Direct Row-Projection Method For Sparse Linear Systems. *Ph.D. thesis, Department of Mathematics, North Carolina State University, Raleigh, NC.* (1993).

[12] Benzi M. Preconditioning techniques for large linear systems: A survey . *J. Comput. Phys., 182* (2002).

[13] Benzi M. and C. D. Meyer. A direct projection method for sparse linear systems. *SIAM J. Sci. Comput. 16*.

[14] Benzi M. and M. Tuma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput. 19*.

[15] Benzi M., C. D. Meyer, and M. Tuma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput. 17* (1995).

[16] BLAS technical forum. www.netlib.org/utk/papers/blast-forum.html. .

[17] Bleck R. Finite difference equations in generalized vertical coordinates . Part I. Total energy conservation. *Beiträge zur Physik der Atmosphäre 51* (1978).

[18] Bolz J., Farmer I. et all. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Trans. Graph. (Proceedings of ACM SIGGRAPH) 22(3).* (2003).

[19] Bridson R. and W.-P. Tang. Refining an approximate inverse. *J. Comput. Appl. Math. 22* (2000).

[20] Bruaset A. M. *A Survey of Preconditioned Iterative Methods.* (Longman Scientific & Technical, Harlow, UK, 1995.

[21] Bryan, F.O., and W.R. Holland. A high resolution simulation of the wind and thermohaline-driven circulation in the North Atlantic ocean. In: P. Muller and D. Anderson (Eds.), Parametrization of Small-Scale Processes. *Proceedings Aha Huliko'a Hawaiian* .

[22] Bryan, K. A numerical method for the study of the circulation of the world ocean. *J. Comput. Phys., 4,* (1969).

[23] Bryan, K., and M.D. Cox,. An approximate equation of state for numerical models of the ocean circulation. *J. Phys. Oceanogr., 2,* (1972).

[24] Byckling M. and M. Huhtanen,. Approximate Factoring of the Inverse. *http://math.tkk.fi/¡≪mhuhtane/* (2002).

[25] Chen C., Beardsley R. C. An unstructured grid, finite-volume coastal ocean model: FVCOM user manual. *SMAST/UMASSD Technical Report* (2004).

[26] Cox, M.D.,. An eddy resolving model of the ventilated thermocline. *J. Phys. Oceanogr., 15* .

[27] Cox, M.D.,. A primitive equation three-dimensional model of the ocean. *Tech. Rep. 1. GFDL Ocean Group, Princeton University* (1984).

[28] Cuomo S. and A. Marasco,. A numerical approach to nonlinear two-point boundary value problems for ODEs. *Computer and Mathematics with Applications* (2008).

[29] Cuomo S. and L.D'Amore e A.Murli,. Error analysis of a Collocation method for numerically inverting a Laplace transform in case of real samples. *Journal of Computational and Applied Mathematics Volume 210, Issues 1-2, 31* (2007).

[30] Cuomo S. and L.D'Amore e A.Murli,. Numerical regularization of a real inversion formula based on the Laplace transform's

eigenfunction expansion of the inverse function. *Inverse Problem 23* (2007).

[31] Cuomo S. and L.D'Amore, M. Rizzardi A.Murli. A Modification of Weeks' Method for Numerical Inversion of the Laplace Transform in the Real Case Based on Automatic Differentiation. *Lecture Notes in Computational Science and Engineering, 2008, Volume 64* (2008).

[32] Cusp Library. . *http://code.google.com/p/cusp-library/* (2009).

[33] Demko S., W. F. Moss, and P. W. Smith,. Decay rates for inverses of band matrices. *Math.Comp., 43* .

[34] Dongarra J. et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[35] Drange H, Simonsen K. Formulation of Air Sea Fluxes in the ESOP2 Version of MICOM. *Technical Report 125 Nansen Environmental and Remote Sensing Center* (1996).

[36] Druskin V., A. Greenbaum et all. Using nonorthogonal Lanczos vectors in the computation of matrix functions. *SIAM J. Sci. Comp., 19* (1998).

[37] Druskin V. and L. A. Knizhnerman. Error bounds in the simple Lanczos procedure for computing functions of symmetric matrices and eigenvalues. *Comput. Math. Math. Phys. 31* (1991).

[38] Dukowicz, J.K., and R.D. Smith,. Implicit free surface method for the Bryan-Cox-Semtner model. *J. Geophys. Res., 99,* (1993).

[39] Dukowicz, J.K., R.D. Smith, and R.C. Malone,. A reformulation and implementation of the Bryan-Cox-Semtner ocean model on the connection machine. *J. Atmos. Ocean Technol., 2,* (1993).

[40] E. A. Lipitakis and D. J. Evans. Explicit semi-direct methods based on approximate inverse matrix techniques for solving boundary-value problems on parallel processors. *Math. Comput.Simulation, 29* (1987).

[41] Epicoco I., S.Mocavero, E.Scoccimarro and G.Aloisio. ORCA025: Performance Analysis on Scalar Architecture. *CMCC Research Paper N. 50*.

[42] Faddeev D. K. and V. N. Faddeeva. *Computational Methods of Linear Algebra*. W. H. Freeman and Co., San Francisco, London, 1963.

[43] Farina R., S. Cuomo and M. Chinnici . Numerical Remarks on the Preconditioned Conjugate Gradient of the Ocean Dynamics Model OPA-NEMO. *CMMSE* .

[44] Farina R., S.Cuomo and P.De Michele . A CUBLAS-CUDA Implementation of PCG Method of an Ocean Circulation Model. *AIP Conf. Proc. September 14, Volume 1389*.

[45] Farina R., S.Cuomo, P.De Michele and M. Chinnici. Inverse Preconditioning Techniques on a GPUs Architecture in Global Ocean Models. *Euro Siam proceedings* (2011).

[46] Fatahalian K., J. Sugerman, and P. Hanrahan. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2004).

[47] Fram group. An eddy-resolving model of the Southern Ocean. *EOS, Trans. Amer. Geophys. Union 72*.

[48] Gill A. E. *Atmosphere-Ocean Dynamics*. Academic Press, San Diego, 1982.

[49] Golub G. and C.Van Loan. *Matrix Computation*. The Johns Hopkins University Press .

[50] Greenbaum A. Behavior of slightly perturbed Lanczos and Conjugate Gradient recurrences, . *Linear Algebra Appl., 113* (1989).

[51] Greenbaum A. *Iterative Methods for Solving Linear Systems*. Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 1997.

[52] Greenbaum A. and Z. STRAKOS. Predicting the behavior of finite precision Lanczos and Conjugate Gradient computations . *SIAM J. Matrix Anal. Appl.* (1992).

[53] Grieco G. and S.Masina. Implementation of NEMO-OPA in Configuration ORCA-R025. *CMCC Research Paper N. 72*.

[54] Gries M., Boning C., Bryan F.O. et al. Developments in ocean climate modelling. *Ocean Modelling Volume 2* (2000).

[55] Griffies S. M. Elements of MOM4p1, GFDL Ocean Group Tech. Rep. *Geophys. Fluid Dyn. Lab., NOAA, Princeton, N. J.* .

[56] Griffies S. M. et all. The MOM manual. *Geophysical Fluid Dynamics Laboratory Tech. Rep., Princeton* (1999).

[57] Grote M.J. and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J.Sci. Comput. 18* (1997).

[58] Haidvogel, D.B., and Beckmann A. Numerical modeling of the c oastal ocean. *In: K.H. Brink and A.R. Robinson (Eds.), The Sea, Vol. 10,* (1998).

[59] Hallberg R. . The HIM User's Manual . *NOAA Geophysical Fluid Dynamics Laboratory, http://www.gfdl.noaa.gov/him-user-manual* (2002).

[60] Haltiner G. J. and R. T. Williams. Numerical prediction and dynamic meteorology. *John Wiley Sons Eds., second edition* (2004).

[61] Hestenes M.R. and E. Stiefel. Methods of Conjugate Gradient for Solving Linear Systems. *J.Res. Nat. Bur. Standars,49* (1952).

[62] Higdon R. Numerical modelling of ocean circulation. *Acta Numerica, Cambridge University Press* (2006).

[63] Householder A. S. *The Theory of Matrices in Numerical Analysis.* Blaisdell Publishing Co., New York, 1964.

[64] Iskandarani, M., Haidvogel, D.B., Boyd, J.P. A staggered spectral model with application to the oceanic shallow water equations. *Int. J. Numer. Meth. Fluids 20* (1995).

[65] Janna C., Ferronato M. e Gambolati G. A Block FSAI-ILU Parallel Preconditioner for Symmetric Positive Definit Linear Systems. *SIAM J. SCI. COMPUT Vol. 32, No. 5* (2010).

[66] Kara C. et all. Sea surface height predictions from the global Navy Coastal Ocean Model (NCOM). *J. Atmos. Oceanic Tech., 21,* .

[67] Kelley C.T. . Iterative Methos for Linear and Nonlinear Equation. *SIAM Philadelphia* (1995).

[68] Kershaw David S. The incomplete Cholesky conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics, Volume 26.*.

[69] Killworth, P.D., D. Stainforth, D.J. Webb, and S.M. Paterson,. The development of a free-surface Bryan-Cox-Semtner ocean model. *J. Phys. Oceanogr., 21,*.

[70] Kolotilina L. Yu. and A. Yu. Yeremin. Factorized sparse approximate inverse preconditioning I. Theory,. *SIAM J. Matrix Anal. Appl., 14* (1993).

[71] Langlois, G. Miami Isopycnic Coordinate Ocean Model (MICOM). User's manual . *Revised and translated by D. Brydon, R. Bleck, and S. Dean. [Available online from http:// www.acl.lanl.gov/CHAMMP/micom.html.]* (1997).

[72] Lehmann, A. A three-dimensional baroclinic eddy-resolving model of the Baltic Sea. *Tellus, 47A*.

[73] Lin C.J. and J. J. More,. Incomplete Cholesky factorizations with limited memory. *SIAM J.Sci. Comput., 21* (1999).

[74] Madec G. . NEMO Reference Manual, Ocean Dynamics Component NEMO-OPA,. *Institut Pierre-Simon Laplace (IPSL),France.* (2009).

[75] Manteuffel T. A. An incomplete factorization technique for positive definite linear systems. *Math. Comp., 34* (1980).

[76] Meijerink J. A. and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp., 31* (1977).

[77] Mellor, G. L., . Users guide for a three-dimensional, primitive equation, numerical ocean model POM. *Prog. in Atmos. and Ocean. Sci, Princeton University* (2003).

[78] Meurant G. A review of the inverse of symmetric tridiagonal and block tridiagonal matrices. *SIAM J. Matrix Anal. Appl., 13* .

[79] NEMO Home page. . *http://www.nemo-ocean.eu/*.

[80] NVIDIA. CUBLAS Library.,http://developer.download.nvidia.com. , .

[81] NVIDIA. CUDA Programming Guide 2.0.,http://www.nvidia.com. .

[82] Nvidia. Fermi GPU architecture, http://www.nvidia.com. (2008).

[83] Paige C.C.and M. A. Saunders. Solution of Sparse Indefinite Systems Of Linear Equations. *SIAM, J. of Numerical Analysis 11.* (1974).

[84] Pain CC., MD Piggott, AJH Goddard, F Fang. Three-dimensional unstructured mesh ocean modelling. *Ocean Modelling, Elsevier* (2005).

[85] Penven, P., Tan, T.,. ROMSTOOLS user's guide. *http://www. brest. ird. fr/roms tools. Tech. rep., IRD.* (2007).

[86] Pierce D. W. The Hybrid Coupled Model, Version 3: Technical Notes. *Institution of Oceanography La Jolla*.

[87] Ricciardi G.,D.Durante,. *Elemeti di fluidodinamica*. Springer, 2006.

[88] Robert, A. J. The integration of lower order spectral form of the primitive meteorological , 1966:. J. Meteor. Soc. Japan, 44, 237. equation. *J. Meteor. Soc. Japan, 44, 237* (1966).

[89] Robinson A. Harvard Ocean Prediction System (HOPS) . *http://oceans.deas.harvard.edu/HOPS/HOPS.html.* ((2001-2004)).

[90] Saad Y. *Iterative Methods for Sparse Linear Systems 2nd.* Society for Industrial and Applied Mathematics Philadelphia, 2003.

[91] Semtner A. Introduction to a numerical method for the study of the circulation of the world ocean. *Journal of Computational Physics* (1997).

[92] Semtner, A.J., and R.M. Chervin,. Ocean general circulation from a global eddy-resolving model. *J. Geophys. Res., 97*.

[93] Shchepetkin, A. F., McWilliams, J. C.,. The regional oceanic mod- eling system (ROMS): a split-explicit, free-surface, topography-following- coordinate oceanic model. *Ocean Model. 9* (2005).

[94] Smith R.D. and Gent P. Reference manual for the Parallel Ocean Program (POP). *Los Alamos Technical Report* (1999).

[95] Strikwerda J. C. Finite Difference Schemes and Partial Differential Equations. *Pacific Grove, CA: Wadsworth & Brooks/Cole Advanced Books & Software*.

[96] Tber L., Hascoët A., Vidard B. et all. Building the Tangent and Adjoint codes of the Ocean General Circulation Model OPA with the Automatic Differentiation tool TAPENADE. *Tech. Rep. INRIA* (2007).

[97] Trefethen L. N., Bau D. . Numerical Linear Algebra. *SIAM, Philadelphia, 1997* (1997).

[98] Treguier, A.M., J.K. Dukowicz, and K. Bryan. Properties of nonuniform grids used in ocean general circulation model. *J. Geophys. Res. ., 101*.

[99] Trites, R. W. e Garrett, C. J. R. Physical oceanography of the Quoddy Region. *In Marine and Coastal Systems of the Quoddy Region new brunswick (thomas, m. l. h., ed.). canadian special publication of fisheries and aquatic sciences* (1983).

[100] Varga R.S.,. Factorization and normalized iterative methods, Boundary problems in differential equations,. *R.E. Langer, ed., University of Wisconsin Press, Madison,* (1960).

[101] Wallcraft, A.J., A.B. Kara, H.E. Hurlburt, and P.A. Rochford,. The NRL Layered Ocean Model (NLOM) with an embedded mixed layer sub-model: Formulation and tuning. . *J. Atmos. Ocean. Technol., 20* (2003).

[102] Winton, M., R. Hallberg, and A. Gnanadesikan. Simulation of density-driven frictinal downslope flow in z-coordinate ocean models. *J. Phys. Oceanogr., 28*.

[103] Young DL., CC.Tsai, CW. Chen, CM. Fan. The method of fundamental solutions and condition number analysis for inverse problems of Laplace equation. *Computers and Mathematic with Applications*.