Risoluzione di Sistemi Lineari di Grandi Dimensioni Utilizzando Calcolo Parallelo su Cluster HPC

Dimonte Giuseppe
Università degli Studi di Parma
giuseppe.dimonte@studenti.unipr.it
Matricola: 367431

Abstract

Questo progetto descrive l'implementazione e l'esecuzione di algoritmi paralleli per la risoluzione di sistemi lineari di grandi dimensioni utilizzando il metodo di Gauss-Seidel su un cluster di calcolo ad alte prestazioni (HPC) presso l'Università di Parma (UniPr). Vengono esplorati tre approcci differenti: MPI, OpenMP e CUDA, per sfruttare al meglio le risorse del cluster HPC.

Contents

1 Introduzione

La risoluzione di sistemi lineari di grandi dimensioni è una delle problematiche fondamentali nel campo del calcolo scientifico. Questo progetto mira a implementare il metodo di Gauss-Seidel in modo parallelo utilizzando tre differenti tecnologie: MPI per il calcolo distribuito, OpenMP per il calcolo parallelo su CPU e CUDA per il calcolo parallelo su GPU.

2 Metodo di Gauss-Seidel

Il metodo di Gauss-Seidel è un algoritmo utilizzato per risolvere sistemi di equazioni lineari. È un tipo di metodo iterativo che converge gradualmente alla soluzione del sistema. Questo metodo è particolarmente utile quando il sistema non è simmetrico o quando si ha bisogno di risolvere sistemi di grandi dimensioni.

L'idea di base del metodo di Gauss-Seidel è di utilizzare le soluzioni approssimate dei valori delle variabili durante l'iterazione stessa, piuttosto che aspettare fino alla fine di ogni iterazione. Inizia con una soluzione iniziale e continua a migliorarla fino a quando non si raggiunge una precisione desiderata.

L'algoritmo procede iterativamente calcolando il valore di ciascuna variabile del sistema di equazioni utilizzando i valori correnti delle altre variabili. Questo processo continua fino a quando i valori delle variabili non cambiano significativamente tra un'iterazione e la successiva, ovvero fino a quando il sistema converge verso la soluzione.

Un vantaggio del metodo di Gauss-Seidel è che può convergere più rapidamente rispetto ad altri metodi iterativi come il metodo di Jacobi, poiché utilizza le informazioni più recenti disponibili durante il processo di iterazione.

Tuttavia, è importante notare che il metodo di Gauss-Seidel non è garantito di convergere per tutti i sistemi di equazioni lineari. Alcuni sistemi possono non convergere o possono richiedere un numero elevato di iterazioni per raggiungere una soluzione accettabile. Inoltre, la convergenza può dipendere dalla scelta della soluzione iniziale e da altri fattori. Pertanto, è importante monitorare la convergenza del metodo e, se necessario, apportare modifiche alla strategia di iterazione.

3 Implementazione del Metodo di Gauss-Seidel con CUDA, MPI e OpenMP

Il metodo di Gauss-Seidel può essere parallelizzato efficacemente utilizzando diverse tecnologie, tra cui CUDA per la programmazione parallela su GPU, MPI per la comunicazione tra processi e OpenMP per il threading parallelo su CPU.

3.1 CUDA

CUDA è una piattaforma di calcolo parallelo sviluppata da NVIDIA per sfruttare la potenza di elaborazione delle GPU. Nell'implementazione del metodo di Gauss-Seidel con CUDA, ogni iterazione del metodo può essere eseguita in parallelo sui thread della GPU, consentendo una rapida convergenza. Le matrici e i vettori possono essere memorizzati nella memoria globale della GPU e i kernel CUDA possono essere utilizzati per eseguire i calcoli paralleli.

3.2 MPI

MPI (Message Passing Interface) è uno standard per la comunicazione parallela su sistemi distribuiti e cluster. Nell'implementazione del metodo di Gauss-Seidel con MPI, i processi possono essere distribuiti su più nodi e comunicare tra loro per scambiare dati e sincronizzare le iterazioni del metodo. Ogni processo può essere responsabile di una porzione dei dati del sistema di equazioni, condividendo e aggiornando i risultati parziali con gli altri processi tramite operazioni di comunicazione MPI.

3.3 OpenMP

OpenMP è una API per la programmazione parallela su sistemi con memoria condivisa, come i sistemi multiprocessore e i nodi di calcolo con più core. Nell'implementazione del metodo di Gauss-Seidel con OpenMP, le iterazioni del metodo possono essere parallelizzate utilizzando direttive di compilazione per creare thread paralleli che lavorano su parti diverse dei dati del sistema di equazioni. Le direttive di OpenMP possono essere utilizzate per gestire la sincronizzazione tra i thread e garantire l'accesso sicuro e concorrente ai dati condivisi.

3.4 Descrizione dei Vantaggi

3.4.1 MPI

- **Scalabilità**: Ideale per sistemi con un gran numero di nodi, come supercomputer e cluster di grandi dimensioni.
- **Flessibilità**: Funziona su una varietà di architetture, incluso cluster eterogenei.
- **Efficienza**: Comunicazione diretta tra processi, riducendo la latenza.
- **Portabilità**: Standard ampiamente adottato disponibile su molte piattaforme.

3.4.2 OpenMP

- **Facilità d'uso**: Parallelizzazione semplice tramite direttive di compilazione.
- **Efficienza in memoria condivisa**: Ideale per sistemi con memoria condivisa come workstation multi-core.
- **Flessibilità**: Controllo granulare sulla parallelizzazione e ottimizzazione delle prestazioni.
- **Compatibilità**: Funziona con codice C, C++ e Fortran esistente.

3.4.3 CUDA

- **Potenza di calcolo**: Sfrutta migliaia di core GPU per un'enorme accelerazione del calcolo.
- **Efficienza energetica**: GPU ottimizzate per calcoli intensivi con consumo energetico ridotto.
- **Librerie ottimizzate**: Supporto per librerie matematiche e scientifiche altamente ottimizzate.
- **Parallelismo fine**: Capacità di gestire migliaia di thread simultaneamente per calcoli complessi.

3.5 Considerazioni finali

Le tecnologie CUDA, MPI e OpenMP offrono approcci diversi per parallelizzare e accelerare l'implementazione del metodo di Gauss-Seidel su architetture moderne. La scelta della tecnologia dipenderà dalle specifiche esigenze dell'applicazione, dalle caratteristiche dell'hardware disponibile e dalla complessità del sistema di equazioni da risolvere.

4 Generazione del Dataset

Per testare gli algoritmi, è stato generato un dataset sintetico utilizzando uno script Python. Il dataset consiste in una matrice A di grandi dimensioni e un vettore b. La matrice A è stata generata in modo tale da essere diagonale dominante per garantire la convergenza del metodo di Gauss-Seidel.

- **Generazione della matrice** A: La matrice è generata casualmente e resa diagonale dominante, dove ogni elemento diagonale è maggiore della somma degli elementi non diagonali nella stessa riga.
- **Generazione del vettore** b: Il vettore è generato casualmente.
- **Salvataggio dei dati**: La matrice A e il vettore b sono salvati in file di testo per essere letti dagli algoritmi implementati.

4.1 Script Python

Lo script Python utilizzato per la generazione del dataset è il seguente:

```
import numpy as np
def generate_dataset(N, filename_A='matrix_A.txt'
    , filename_b='vector_b.txt'):
    # Genera una matrice casuale NxN
    A = np.random.rand(N, N)
    # Rendere A diagonale dominante
    for i in range(N):
        A[i, i] = sum(np.abs(A[i])) + 1
    # Genera un vettore casuale Nx1
    b = np.random.rand(N)
    # Salva la matrice A e il vettore b in file di testo
    np.savetxt(filename_A, A, fmt='%.6f')
    np.savetxt(filename_b, b, fmt='%.6f')
# Esempio di utilizzo
generate_dataset(1000)
```

In questo script, la funzione `generate_dataset` prende come parametro il numero N di righe e colonne della matrice A, e genera un dataset con una matrice A di dimensioni $N \times N$ e un vettore b di dimensioni N. Nell'esempio di utilizzo, la funzione è chiamata con $N = 1000$, generando così una matrice di 1000 righe e 1000 colonne e un vettore di 1000 elementi.

5 Implementazione

5.1 Implementazione OpenMP

L'implementazione OpenMP sfrutta la parallelizzazione su CPU tramite direttive di compilazione. I passaggi principali sono i seguenti:

- **Inizializzazione dei dati**: La matrice A e i vettori b e x vengono inizializzati.
- **Iterazioni di Gauss-Seidel**: Utilizzando la direttiva `#pragma omp parallel for`, il ciclo principale dell'algoritmo viene parallelizzato. Ogni thread calcola i nuovi valori del vettore x per un sottoinsieme delle righe della matrice.
- **Controllo della convergenza**: La convergenza viene verificata calcolando la norma del residuo e interrompendo l'iterazione se la norma è inferiore a una soglia di tolleranza.
- **Pulizia**: Alla fine dell'algoritmo, la memoria allocata viene liberata.

Il codice completo OpenMP del metodo di Gauss-Seidel può essere trovato al seguente link: Codice OpenMP Gauss-Seidel

5.2 Implementazione MPI

L'implementazione MPI del metodo di Gauss-Seidel prevede l'utilizzo della libreria MPI per la comunicazione tra processi. Di seguito vengono descritti i passaggi principali dell'algoritmo.

- **Inizializzazione MPI**: Viene inizializzato l'ambiente MPI con `MPI_Init`, ottenendo il rank del processo corrente e il numero totale di processi con `MPI_Comm_rank` e `MPI_Comm_size` rispettivamente.

- **Divisione del lavoro**: La matrice A e il vettore b vengono suddivisi in parti uguali tra i processi utilizzando `MPI_Scatter`. Ogni processo riceve un sottoinsieme delle righe della matrice e del vettore. Le righe della matrice e gli elementi del vettore vengono distribuiti in blocchi contigui.

- **Iterazioni di Gauss-Seidel**: Per ogni iterazione, ogni processo calcola i nuovi valori del proprio sottoinsieme del vettore soluzione x. La somma dei prodotti degli elementi non diagonali viene calcolata e sottratta dal termine noto, poi divisa per l'elemento diagonale. Ogni processo esegue questa operazione per le righe che gli sono state assegnate.

- **Comunicazione tra processi**: I nuovi valori calcolati dai processi vengono comunicati a tutti gli altri processi utilizzando `MPI_Allgather`, in modo che ogni processo abbia una copia aggiornata del vettore soluzione x. Questo garantisce che tutti i processi abbiano sempre una vista coerente dei valori di x.

- **Controllo della convergenza**: Viene calcolata la norma del residuo locale per ogni processo e i risultati vengono ridotti a una somma globale utilizzando `MPI_Allreduce`. Se la norma globale è al di sotto di una certa soglia di tolleranza, l'iterazione si interrompe. Questo controllo è effettuato in modo sincrono tra tutti i processi.

- **Pulizia e finalizzazione**: Alla fine dell'algoritmo, la memoria allocata viene liberata e l'ambiente MPI viene chiuso con `MPI_Finalize`. Ogni processo esegue questa operazione per garantire una corretta chiusura dell'ambiente MPI.

Il codice completo MPI del metodo di Gauss-Seidel può essere trovato al seguente link: Codice MPI Gauss-Seidel

5.3 Implementazione CUDA

L'implementazione CUDA dell'algoritmo di Gauss-Seidel sfrutta la parallelizzazione su GPU per accelerare il calcolo. I passaggi principali sono i seguenti:

- **Allocazione e copia dei dati**: La matrice A e i vettori b e x vengono trasferiti dalla memoria dell'host (CPU) alla memoria del dispositivo (GPU) utilizzando le funzioni `cudaMalloc` e `cudaMemcpy`.

- **Kernel di Gauss-Seidel**: Un kernel CUDA viene lanciato per calcolare i nuovi valori del vettore x in parallelo su più thread. Ogni thread calcola il valore per un singolo elemento del vettore.
- **Iterazioni**: Il kernel viene lanciato ripetutamente per il numero massimo di iterazioni o fino al raggiungimento della convergenza.
- **Copia dei risultati**: I risultati vengono copiati dalla memoria del dispositivo alla memoria dell'host utilizzando la funzione `cudaMemcpy`.
- **Pulizia**: La memoria allocata sulla GPU viene liberata utilizzando la funzione `cudaFree`.
- **Misurazione del tempo e report**: Il tempo di esecuzione viene misurato prima e dopo l'esecuzione del kernel utilizzando funzioni specifiche. Viene generato un report che include il tempo di esecuzione, il numero di thread utilizzati, e la GPU impiegata.

Il codice completo CUDA del metodo di Gauss-Seidel può essere trovato al seguente link: Codice CUDA Gauss-Seidel

6 Configurazione ed Esecuzione su HPC

Per eseguire i programmi parallelizzati OpenMP, MPI e CUDA su un cluster HPC, è necessario configurare un file SLURM. Questo file specifica i dettagli del job, come il nome, il tempo massimo di esecuzione, il numero di task, il numero di CPU per task, la partizione e la qualità del servizio (QoS). Inoltre, carica i moduli necessari per la compilazione e l'esecuzione dei programmi e gestisce la misurazione del tempo di esecuzione e la generazione del report.

6.1 Configurazione del File SLURM

Il file SLURM configura e gestisce l'esecuzione dei programmi parallelizzati MPI, OpenMP E CUDA su un cluster HPC. Ecco i passaggi principali e le configurazioni:

- **Configurazione del Job**: Specifica il nome del job, i file di output e di errore, il tempo massimo di esecuzione, quanti processi e quanti core nel caso di MPI, quanti thread e quanti core nel caso di OpenMP e quanti kernel thread nel caso di CUDA e quale GPU è stata usata.

- **Caricamento dei Moduli**: Carica i moduli necessari per la compilazione e l'esecuzione del programma, come il compilatore GCC per OpenMPI , il compilatore mpicc per MPI e il compilatore nvcc per CUDA se necessario.
- **Compilazione del Programma**: Compila OpenMPI con `GCC`, compila MPI con `mpicc` e compila CUDA con `nvcc`.
- **Esecuzione del Programma**: Esegue il programma compilato utilizzando `srun`, che gestisce l'esecuzione del programma MPI sui nodi del cluster.
- **Generazione del Report**: Misura il tempo di esecuzione e scrive il report nel file di output, includendo il tempo di esecuzione in secondi, il numero di processi MPI, thread e core oer OpneMP e kernel thread nel caso di CUDA e quale GPU è usata e i moduli caricati.

7 Confronto dei Risultati

7.1 CUDA

Table 1: CUDA

Implemen	T di Es (s)	GPU Utilizzata	N Thread Blocco	N di Blocchi
CUDA	0.424990	Tesla P100-PCIE-12GB	256	4

Analisi dei Risultati

L'implementazione CUDA ha ottenuto il tempo di esecuzione più basso tra tutte le implementazioni, con soli 0.424990 secondi. Questo è dovuto all'efficace utilizzo della potenza computazionale della GPU Tesla P100-PCIE-12GB, sfruttando 256 thread per blocco distribuiti su 4 blocchi. La parallela parallelizzazione su GPU si è dimostrata molto efficace per questo tipo di calcolo intensivo.

7.2 OpenMP

Analisi dei Risultati

L'implementazione OpenMP ha mostrato un tempo di esecuzione di 0.957124292 secondi, utilizzando 4 thread su 4 core. Sebbene l'efficienza sia buona, risulta

Table 2: OpenMP

Implementazione	T di Esecuzione (s)	N di Thread OpenMP	N di Core Utilizzati
OpenMP	0.957124292	4	4

essere più lenta rispetto alla versione CUDA. Questo suggerisce che la parallelizzazione su CPU tramite OpenMP è meno performante rispetto alla GPU per questo tipo di algoritmo.

7.3 MPI

Table 3: MPI

Implementazione	T di Esecuzione (s)	N di Processi MPI	N di Core per Processo
MPI	3	4	1

Analisi dei Risultati

L'implementazione MPI ha registrato un tempo di esecuzione di 3 secondi, utilizzando 4 processi con 1 core ciascuno. Questo è significativamente più lento rispetto a entrambe le altre implementazioni. La divisione del lavoro tra più processi su nodi separati può introdurre un overhead significativo nella comunicazione e nella coordinazione, influenzando le prestazioni complessive.

8 Conclusione

Questo progetto ha dimostrato come l'algoritmo di Gauss-Seidel possa essere parallelizzato utilizzando tre diverse tecnologie: MPI, OpenMP e CUDA. Ogni tecnologia offre vantaggi specifici a seconda dell'architettura hardware disponibile e delle caratteristiche del problema da risolvere. Utilizzando il cluster HPC dell'Università di Parma, è stato possibile sfruttare al massimo le capacità di calcolo parallelo per risolvere sistemi lineari di grandi dimensioni in modo efficiente.

A Struttura del Progetto

Il progetto è organizzato nelle seguenti directory:

- `mpi/`
  - `gauss_seidel_mpi.c` - Codice sorgente C per l'implementazione MPI.
  - `gauss_seidel_mpi.slurm` - Script SLURM per eseguire il codice MPI sul cluster.
  - `gauss_seidel_mpi_output.txt` - File di output generato dall'esecuzione del codice MPI.
- `openmp/`
  - `gauss_seidel_omp.c` - Codice sorgente C per l'implementazione OpenMP.
  - `gauss_seidel_omp.slurm` - Script SLURM per eseguire il codice OpenMP sul cluster.
  - `gauss_seidel_omp_output.txt` - File di output generato dall'esecuzione del codice OpenMP.
- `cuda/`
  - `gauss_seidel_cuda.cu` - Codice sorgente CUDA per l'implementazione su GPU.
  - `gauss_seidel_cuda.slurm` - Script SLURM per eseguire il codice CUDA sul cluster.
  - `gauss_seidel_cuda_output.txt` - File di output generato dall'esecuzione del codice CUDA.
- `report/`
  - `relazione_progetto.pdf` - Relazione realizzata in LaTeX che descrive il progetto.

B Requisiti

- Accesso a un cluster HPC presso l'Università di Parma (UniPr).
- Compilatori C/C++ con supporto per MPI, OpenMP e CUDA.
- Ambiente di scripting SLURM per la gestione dei job sul cluster.

C Istruzioni per l'Esecuzione

C.1 Passaggi Generali

1. **Accedere al Cluster HPC**: Utilizzare le credenziali fornite per accedere al cluster HPC di UniPr.

   ```
   ssh nome.utente@login.hpc.unipr.it
   ```

2. **Utilizzo di WinSCP per il Trasferimento di File**:WinSCP è un client SFTP e FTP per Windows che permette di trasferire file in modo sicuro tra un computer locale e un server remoto. Di seguito sono riportati i passaggi per installare e utilizzare WinSCP.

3. **Installazione di WinSCP**: Per installare WinSCP, seguire questi passaggi:

   (a) Visitare il sito web ufficiale di WinSCP: `https://winscp.net/`.

   (b) Fare clic sul pulsante `Download Now` per scaricare l'installer.

   (c) Eseguire il file scaricato e seguire le istruzioni dell'installazione guidata.

   (d) Configurazione di WinSCP
   Una volta installato WinSCP, è possibile configurarlo per connettersi al cluster HPC dell'Università di Parma (UniPr). Seguire questi passaggi:

      i. Avviare WinSCP.

      ii. Nella schermata di login, inserire le seguenti informazioni:
      - **Protocollo di file**: FSTP
      - **Nome host**: l'indirizzo del server del cluster HPC (fornito dall'Università)
      - **Numero di porta**: 22 (default per FSTP)
      - **Nome utente**: il proprio username per il cluster HPC
      - **Password**: la propria password per il cluster HPC

      iii. Fare clic su `Salva` se si desidera salvare queste informazioni per future connessioni.

      iv. Fare clic su `Login` per connettersi al server.

   (e) Trasferimento di File con WinSCP: Dopo essersi connessi al server, è possibile trasferire file tra il computer locale e il server remoto. Ecco come fare:

i. Nella finestra principale di WinSCP, la parte sinistra rappresenta il file system locale, mentre la parte destra rappresenta il file system remoto.

ii. Per trasferire file dal computer locale al server remoto, selezionare i file desiderati nella parte sinistra e trascinarli nella posizione desiderata nella parte destra.

iii. Per trasferire file dal server remoto al computer locale, selezionare i file desiderati nella parte destra e trascinarli nella posizione desiderata nella parte sinistra.

iv. È anche possibile utilizzare il menu contestuale (clic destro) per opzioni avanzate come la modifica dei permessi dei file, la creazione di nuove directory, la sincronizzazione delle cartelle, e altro.

4. **Esempio di Utilizzo per il Progetto**: Per il progetto, utilizzare WinSCP per trasferire i file di output generati dalle implementazioni CUDA, MPI e OpenMP dal cluster HPC al proprio computer locale per ulteriori analisi. Ad esempio:

   (a) Connettersi al cluster HPC utilizzando WinSCP.

   (b) Navigare nella directory contenente i file di output, ad esempio `/mpi/gauss_seidel_mpi_output.txt`, `/openmp/gauss_seidel_omp_output.txt`, e `/cuda/gauss_seidel_cuda_output.txt`.

   (c) Selezionare e trasferire questi file sul proprio computer locale.

   Con WinSCP, il trasferimento di file tra il proprio computer e il cluster HPC diventa semplice e sicuro, permettendo di analizzare facilmente i risultati generati dalle diverse implementazioni del metodo di Gauss-Seidel.

5. **Conversione delle linee di fine file**: Per convertire i file in formato UNIX.

```
dos2unix gauss_seidel_omp.slurm
dos2unix gauss_seidel_mpi.slurm
dos2unix gauss_seidel_cuda.slurm
```

6. **Verifica le versioni disponibili dei moduli**: Usa module spider openmpi per elencare tutte le versioni di openmpi disponibili:

```
module spider openmpi
```

7. **Caricare i Moduli Necessari**: Prima di compilare ed eseguire i programmi, assicurarsi di caricare i moduli necessari per MPI, OpenMP e CUDA. Esempio:

```
module purge
module load gcc
module load gnu8/8.3.0
module load openmpi4/4.1.1
module load cuda/11.6.0
```

8. **Verifica dei Risultati**:

```
cat gauss_seidel_cuda.out
cat gauss_seidel_cuda.err

cat gauss_seidel_omp.out
cat gauss_seidel_omp.err

cat gauss_seidel_mpi.out
cat gauss_seidel_mpi.err
```

9. **Controlla lo stato del job**:Usa squeue per verificare lo stato del job:

```
squeue -u $USER
```

10. **Squeue**: Questo comando mostra una panoramica dello stato di tutti i job attualmente eseguiti nel sistema SLURM. Puoi usare opzioni come -u per visualizzare solo i job dell'utente specificato.

```
squeue -u username
```

11. **Dettagli di un job**: scontrol show job : Questo comando mostra i dettagli di un job specifico, incluso il suo stato e altre informazioni rilevanti.

```
scontrol show job <job_id>
```

12. **Storico dei job**: sacct: Questo comando consente di visualizzare lo storico dei job SLURM eseguiti. Può mostrare informazioni dettagliate sul tempo di esecuzione, l'uso delle risorse, lo stato dei job, e altro ancora.

    ```
    sacct -u username
    ```

13. **Informazioni**: sinfo: Questo comando fornisce informazioni sulle partizioni e sulle risorse disponibili sul cluster.

    ```
    sinfo
    ```

14. **Cancellare i job**: Usa scancel per cancellare i job:

    ```
    scancel -u $USER
    ```

C.2 Generazione del Dataset

Per generare un dataset sintetico, utilizzare lo script Python fornito:

```
python generate_dataset.py
```

C.3 Implementazione MPI

1. **Compilare il Codice MPI**:

   ```
   mpicc -o gauss_seidel_mpi gauss_seidel_mpi.c
   ```

2. **Eseguire il Codice MPI**: Utilizzare lo script SLURM per inviare il job al cluster:

   ```
   cd mpi
   sbatch gauss_seidel_mpi.slurm
   ```

3. **Verificare l'Output**: L'output sarà disponibile nel file `gauss_seidel_mpi_output.txt`.

C.4 Implementazione OpenMP

1. **Compilare il Codice OpenMP**:

   ```
   gcc -fopenmp -o gauss_seidel_omp gauss_seidel_omp.c
   ```

2. **Eseguire il Codice OpenMP**: Utilizzare lo script SLURM per inviare il job al cluster:

   ```
   cd ../openmp
   sbatch gauss_seidel_omp.slurm
   ```

3. **Verificare l'Output**: L'output sarà disponibile nel file `gauss_seidel_omp_output.txt`.

C.5 Implementazione CUDA

1. **Compilare il Codice CUDA**:
   Per usare le P100 si può compilare così:

   ```
   module load cuda
   nvcc src_name.cu arch=compute_60 --o exec_name
   ```

   Per usare le A100 arch=compute_80

   ```
   nvcc -arch=compute_80 -o gauss_seidel_cuda gauss_seidel_cuda.cu
   ```

2. **Eseguire il Codice CUDA**:
   Utilizzare lo script SLURM per inviare il job al cluster:

   ```
   cd ../cuda
   sbatch gauss_seidel_cuda.slurm
   ```

3. **Verificare l'Output**:
   L'output sarà disponibile nel file `gauss_seidel_cuda_output.txt`.

D Passaggi per Compilare ed Eseguire su un Computer con GPU NVIDIA

D.1 Installare i Driver NVIDIA

Assicurati di avere installati i driver NVIDIA appropriati per la tua GPU.

D.2 Installare CUDA Toolkit

Installa il CUDA Toolkit. Puoi scaricarlo dal sito ufficiale di NVIDIA qui.

D.3 Impostare le Variabili d'Ambiente

Dopo aver installato il CUDA Toolkit, imposta le variabili d'ambiente. Tipicamente, aggiungerai queste righe al tuo file `.bashrc` o `.bash_profile`:

```
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Dopo aver aggiunto queste righe, esegui `source /.bashrc` (o `source /.bash_profile`) per applicare le modifiche.

D.4 Aggiungere le Variabili d'Ambiente su Windows

D.4.1 Nella sezione "Variabili di sistema" della finestra "Variabili d'ambiente"

Clicca su "Nuova..." per aggiungere una nuova variabile.

Aggiungi `CUDA_PATH`:

- Nome variabile: `CUDA_PATH`
- Valore variabile: `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.6`

Modifica Path:

- Cerca e seleziona la variabile `Path` nella lista delle variabili di sistema e clicca su "Modifica...".
- Clicca su "Nuovo" e aggiungi il percorso bin di CUDA:

  `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.6\bin`

- Clicca su "Nuovo" e aggiungi il percorso delle librerie di CUDA:

  `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.6\libnvvp`

Aggiungi `LD_LIBRARY_PATH` (opzionale):

- Nome variabile: `LD_LIBRARY_PATH`
- Valore variabile: `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.6\lib\x`

Salva le Modifiche: Dopo aver aggiunto o modificato le variabili, clicca su "OK" per chiudere ogni finestra di dialogo aperta.

Verifica le Variabili d'Ambiente: Apri un nuovo terminale (Prompt dei comandi o PowerShell) e verifica che le variabili siano state impostate correttamente eseguendo i seguenti comandi:

```
echo %CUDA_PATH%
echo %PATH%
echo %LD_LIBRARY_PATH%
```

Assicurati che i percorsi corretti vengano visualizzati.

D.5 Impostare le Variabili d'Ambiente su Linux

Dopo aver installato il CUDA Toolkit, imposta le variabili d'ambiente. Tipicamente, aggiungerai queste righe al tuo file `.bashrc` o `.bash_profile`:

```
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Dopo aver aggiunto queste righe, esegui `source /.bashrc` (o `source /.bash_profile`) per applicare le modifiche.

D.6 Compilare il Codice

Usa `nvcc`, il compilatore di NVIDIA per CUDA, per compilare il tuo codice. Supponendo che il tuo file si chiami `gauss_seidel_cuda.cu`, esegui:

```
nvcc -o gauss_seidel_cuda gauss_seidel_cuda.cu
```

D.7 Eseguire il Programma

Una volta compilato, esegui il programma:

```
./gauss_seidel_cuda
```

E Bibliografia

References

[1] P. S. Pacheco, *An Introduction to Parallel Programming with MPI*, Morgan Kaufmann, 2011.

[2] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, MIT Press, 2008.

[3] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional, 2010.

[4] WinSCP, *Secure Copy Protocol (SCP) Client for Windows*, `https://winscp.net/`

[5] PuTTY, *Free SSH, Telnet, and Rlogin Client*, `https://www.putty.org/`

[6] Python Software Foundation, *Python Programming Language*, `https://www.python.org/`

[7] NVIDIA Corporation, *NVIDIA Developer*, `https://developer.nvidia.com/`

[8] Centro di Calcolo dell'Università di Parma, *HPC Cluster User Guide*, 2024. [Online]. Available: `https://www.hpc.unipr.it/dokuwiki/doku.php?id=calcoloscientifico:userguide`

[9] Il progetto in tutte le sue parti è stato caricato al seguente link: `https://github.com/sh4nk7/Risoluzione-di-Sistemi-Lineari-di-Grandi-Dimensioni`