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

Implementazione e Confronto tra MPI, OpenMP e CUDA

Giuseppe Dimonte
MAT 367431

Università degli Studi di Parma

Abstract

- **Progetto:**
  - Implementazione ed esecuzione di algoritmi paralleli per la risoluzione di sistemi lineari di grandi dimensioni
  - Utilizzo del metodo di Gauss-Seidel su un cluster HPC
  - Tecnologie esplorate: MPI, OpenMP e CUDA

Introduzione

Il metodo di Gauss-Seidel è un algoritmo utilizzato per risolvere sistemi di equazioni lineari.

- Metodo iterativo
- Particolarmente utile per sistemi non simmetrici o di grandi dimensioni

Idea di Base

Utilizza soluzioni approssimate durante l'iterazione stessa.

- Inizia con una soluzione iniziale
- Continua a migliorare fino a raggiungere la precisione desiderata

Procedura - Inizializzazione

1. Scegli un'approssimazione iniziale $x^{(0)}$
2. Imposta un criterio di arresto (es. numero massimo di iterazioni o tolleranza sull'errore)

Procedura - Iterazione

Per ogni equazione i del sistema, aggiorna il valore di x_i:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

Procedura - Convergenza

Continua ad iterare finché la soluzione non soddisfa il criterio di arresto:

$$\|x^{(k+1)} - x^{(k)}\| < \epsilon$$

dove ϵ è la tolleranza prefissata.

Il metodo converge se la matrice A è:

- A dominanza diagonale stretta
- Simmetrica e definita positiva

Esempio

Supponiamo di voler risolvere il sistema:

$$\begin{cases} 3x_1 + x_2 = 5 \\ x_1 + 2x_2 = 5 \end{cases}$$

- Inizializzazione: $x^{(0)} = [0, 0]$

Iterazioni - Esempio

Iterazione 1:

$$x_1^{(1)} = \frac{1}{3}(5 - 0) = \frac{5}{3}$$

$$x_2^{(1)} = \frac{1}{2}(5 - \frac{5}{3}) = \frac{10}{6} \approx 1.67$$

Iterazione 2:

$$x_1^{(2)} = \frac{1}{3}(5 - 1.67) \approx 1.11$$

$$x_2^{(2)} = \frac{1}{2}(5 - 1.11) \approx 1.94$$

Vantaggi e Svantaggi del Metodo Gauss-Seidel

Vantaggi del Metodo

- Può convergere più rapidamente rispetto al metodo di Jacobi
- Semplice da implementare
- Richiede meno memoria rispetto ai metodi diretti

Svantaggi del Metodo

- Non garantisce la convergenza per tutti i sistemi
- Può richiedere un numero elevato di iterazioni
- La convergenza può dipendere dalla scelta della soluzione iniziale

Applicazioni

Utilizzato in ingegneria, fisica e matematica applicata per:

- Risoluzione di grandi sistemi di equazioni sparse

Implementazione con CUDA

- Sfrutta la potenza di elaborazione delle GPU
- Iterazioni parallele sui thread della GPU
- Utilizzo di CUDA per eseguire calcoli paralleli e accelerare la convergenza
- CUDA (Compute Unified Device Architecture) è una piattaforma di calcolo parallelo di NVIDIA.
- Le GPU eseguono operazioni in parallelo su molti thread, accelerando notevolmente il calcolo.
- Implementa algoritmi come il metodo di Gauss-Seidel sfruttando la potenza delle GPU.

Implementazione con MPI

- Standard per la comunicazione parallela su sistemi distribuiti
- Utilizzato per la distribuzione del lavoro tra nodi del cluster
- Comunicazione tra processi per sincronizzare le iterazioni del metodo di Gauss-Seidel
- MPI (Message Passing Interface) facilita la comunicazione tra processi su cluster.
- Essenziale per distribuire il lavoro in modo efficiente tra i nodi di un cluster.
- Sincronizza iterazioni durante il calcolo parallelo del metodo di Gauss-Seidel.

Implementazione con OpenMP

- API per la programmazione parallela su sistemi con memoria condivisa
- Utilizzato per la parallelizzazione su CPU
- Direttive di compilazione per gestire il threading parallelo e ottimizzare le prestazioni
- OpenMP (Open Multi-Processing) semplifica la programmazione parallela su multi-core.
- Ottimizzato per sfruttare le capacità multi-core delle moderne CPU.
- Le direttive di compilazione di OpenMP gestiscono threading parallelo per prestazioni ottimali.

Generazione del Dataset

- La matrice A è generata casualmente e modificata per essere diagonale dominante.
- Il vettore dei termini noti b è creato con valori casuali per rappresentare le soluzioni.
- I dati sono salvati in file di testo per l'input degli algoritmi di risoluzione del sistema.

Configurazione ed Esecuzione su HPC

Per eseguire programmi parallelizzati OpenMP, MPI e CUDA su un cluster HPC, è necessario configurare un file SLURM. Questo file specifica i dettagli del job:

- Nome del job
- Tempo massimo di esecuzione
- Quanti processi e quanti core nel caso di MPI
- Quanti thread e quanti core nel caso di OpenMP
- Quanti kernel thread nel caso di CUDA e quale GPU è stata usata
- Generazione del report

Inoltre, carica i moduli necessari per la compilazione ed esecuzione dei programmi.

Gestione del Job

Il file SLURM ci genera dei report, dai quali ricavo:

- Misurazione del tempo di esecuzione
- Tempo di esecuzione dell'algoritmo sequenziale
- Calcolo della media dei tempi
- Deviazione standard
- Generazione del report
- Speedup

Implementazione della Simulazione

1. Implementa l'algoritmo sequenziale e gli algoritmi (MPI, CUDA, OpenMP) per Gauss-Seidel.
2. Misura il tempo di esecuzione dell'algoritmo sequenziale per un numero fisso di iterazioni.
3. Misura il tempo di esecuzione degli algoritmi per Gauss-Seidel per lo stesso numero di iterazioni.

Calcolo del Tempo Medio

- Calcola il tempo medio di esecuzione per gli algoritmi (sequenziale e paralleli) basato sui 10 risultati ottenuti.

$$\text{Tempo medio di esecuzione} = \frac{1}{10} \sum_{i=1}^{10} T_i$$

dove T_i sono i tempi di esecuzione registrati.

Calcolo della Deviazione Standard

- Calcola la deviazione standard del tempo di esecuzione per entrambi gli algoritmi.

$$\text{Deviazione standard} = \sqrt{\frac{1}{10}\sum_{i=1}^{10}(T_i - \text{Tempo medio})^2}$$

Calcolo dello Speedup

- Lo speedup è il rapporto tra il tempo di esecuzione dell'algoritmo sequenziale e quello degli algoritmi paralleli.

$$\text{Speedup} = \frac{\text{Tempo di esecuzione sequenziale}}{\text{Tempo di esecuzione Algoritmi}}$$

Confronto delle Prestazioni

Il confronto delle prestazioni dei diversi metodi di Gauss-Seidel è riassunto nella Tabella 1. Ogni metodo è stato eseguito su un dataset di dimensione $N = 1000$.

Metodo	**Media (s)**	**Deviazione Std. (s)**	**Speedup**
Gauss-Seidel CUDA	1.780	0.050	2.801
Gauss-Seidel MPI	1.165	0.223	4.27
Gauss-Seidel OpenMP	0.655	0.462	7.56

Tabella: Confronto delle prestazioni dei metodi Gauss-Seidel con dataset di dimensione 1000.

Gauss-Seidel CUDA

Imp	GPU Utilizzata	N Thread Blocco	N di Blocchi
CUDA	Tesla P100-PCIE-12GB	256	4

Tabella: Dettagli delle implementazioni per CUDA

- Utilizzando la GPU
- Tempo medio: 1.780 secondi
- Deviazione standard: 0.050 secondi
- Speedup: circa 2.801 rispetto al metodo sequenziale

Gauss-Seidel MPI

Implementazione	N di Processi MPI	N di Core per Processo
MPI	4	1

Tabella: Dettagli delle implementazioni per MPI

- Utilizzando 4 processi MPI, ognuno con 1 core
- Tempo medio: 1.165 secondi
- Deviazione standard: 0.223 secondi
- Speedup: circa 4.27 rispetto al metodo sequenziale

Gauss-Seidel OpenMP

Implementazione	N di Thread OpenMP	N di Core Utilizzati
OpenMP	4	4

Tabella: Dettagli delle implementazioni per OpenMP

- Utilizzando 4 thread OpenMP su 4 core
- Tempo medio: 0.655 secondi
- Deviazione standard: 0.462 secondi
- Speedup: circa 7.56 rispetto al metodo sequenziale

Conclusioni

Questi passaggi permettono una valutazione dettagliata delle prestazioni degli algoritmi di Gauss-Seidel implementati in CUDA, MPI, OpenMP rispetto alla loro controparte sequenziale.

- Tecnologie diverse offrono approcci diversi per parallelizzare il metodo di Gauss-Seidel
- Scelta della tecnologia dipende dalle esigenze dell'applicazione e dall'hardware disponibile
- MPI, OpenMP e CUDA hanno vantaggi specifici a seconda del contesto