Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Relazione di CALCOLO PARALLELO

BitonicSort VS QuickSort

Autori

Matteo Perin ******

Davide Pistilli 1204880

Flavio Spanò 1197697

Anno Accademico 2018-2019

Indice

Capitolo 1

Descrizione del Problema

Abbiamo implementato un algoritmo di ordinamento dei campioni tramite una versione parallela di Quicksort. I nostri esperimenti computazionali mostrano che Quicksort parallelo risulta essere piú veloce rispetto all'ordinamento del campione fatto in modo iterativo utilizzando un solo Processore. Su 32 processori del Power7 la velocità di Quicksort parallela è più di ..... unità superiori della velocità del Quicksort Iterativo, con tempi di esecuzione più elevati del .....L'algoritmo Quicksort parallelo che abbiamo implementato è una semplice estensione Quicksort, dove l'array di input viene suddiviso in parti uguali a tutti i processori, e ogni processore esegue una serie di Round che permettono di ordinare la parte assegnata ad ogni processore. Una volta finito un round, sincronizza la parte dell'array a lui assegnato, inviando una richiesta non bloccante al proprio root che permettera di proseguire al prossimo round. Queste richieste permetteranno la comunicazione dello stato dei processori e dell'aggiornamento del array di partenza, che a causa della assenza della memoria condivisa ogni processore non sa se altri processori lavorano su stesse locazioni di memoria.

Capitolo 2

DescrizioneAlgoritmo

Quicksort è un algoritmo di ordinamento sequenziale ampiamente ritenuto essere l'algoritmo di ordinamento sequenziale più veloce per un ampio set di input, infatti il termine che tradotto letteralmente in italiano indica ordinamento rapido. È l'algoritmo di ordinamento che ha, nel caso medio, prestazioni migliori tra quelli basati su confronto. È stato ideato da Charles Antony Richard Hoare nel 1961.
È un algoritmo ricorsivo che usa il metodo "Divide and Conquer" per ordinare tutti i valori. Quicksort dal momento che scompone ricorsivamente i dati da processare in sottoprocessi, tale procedura viene chiamata Partition, preleva prima un pivot da una struttura dati, trova la sua posizione nell'elenco in cui deve essere posizionata.
Avremo due possibilità:

- I valori minori del pivot saranno posizionati nella parte sinistra dell'array
- I valori maggiori o uguali saranno posizionati nella parte di destra dell'array

Capitolo 3

Bitonic Sort

Capitolo 4

QuickSort

Supponiamo di disporre un array con N elementi, indicizzato da 0 a N-1 (i_arSize), da ordinare su un multiprocessore a memoria non condivisa, con processori asincroni MAX_PROCESSORS. Ad ogni processore viene assegnato un indice univoco, pid, da 0 a P-1 (_rank).
L'algoritmo parallelo Quicksort presentato è una parallelizzazione del Quicksort. È un algoritmo a 3 + 1 round. I primi 3 round costituiscono il divide la fase e sono eseguiti in modo ricorsivo. L'ultima fase è un ordinamento iterativo dell'algoritmo che i processori eseguono in parallelo.
Le quattro fasi sono:

1. la Partizione parallela dei Dati
2. la Partizione sequenziale dei dati
3. la fase di partizione del processo (fase Ricorsiva del Round 1 e 2)
4. l'Ordinamento iterativo in parallelo

4.1 Round 1: Partizione parallela dei dati

L'algoritmo legge in input una serie di dati come un insieme di blocchi consecutivi di dimensioni i_arSize. ******* i_arSize dipende dalla dimensione del sistema cache di primo livello, ed è selezionata in modo che due blocchi di lunghezza B possano essere contenuti nella cache in corrispondenza di contemporaneamente. Nel nostro sistema in cui la dimensione della cache di primo livello è 16 KB, abbiamo selezionato B = 2048 in modo da essere in grado di adattare contemporaneamente due blocchi di dati nella cache. **********

Consideriamo prima di tutto il caso in cui tutti i valori possono essere divisi in blocchi, nel caso in cui un blocco non dovesse essere riempito completamente, eseguiremo il padding della parte rimante per completare il blocco.
L'intero array può essere visto come una linea di N blocchi di dati; il processore P0 sceglie i blocchi su cui lavorare, dalle due estremità della linea. La prima fase inizia con il processore P 0 , quello con il pid più piccolo, successivamente,il processore P0 invia ad ogni processore il blocco che trova fine del lato sinistro e poi il blocco che trova alla fine del lato destro e usa questi due blocchi insieme con il pivot come input per una funzione chiamata neutralize. La funzione

prende come input due blocchi, ar_left e ar_right e i_pivot, nel nostro caso abbiamo implementato la funzione "PivotChoice" che seleziona 3 valori rand presi nel nostro array e calcola la media del massimo e minimo valore scelto. Una volta trovato il pivot, scambia i valori in leftblock se sono più grandi del pivot con i valori del blocco di destra che sono più piccoli del pivot.
La chiamata del neutralize avrà uno dei seguenti risultati:

- Tutti i valori nel blocco di sinistra saranno inferiori al pivot, in questo caso diciamo quel leftblock è stato neutralizzato
- Tutti i valori nel blocco di destra saranno più grandi o uguali del pivot, diciamo che il blocco di destra è stato neutralizzato
- Sia leftblock che rightblock sono stati neutralizzati allo stesso tempo

Una volta conclusa la prima chiamata di neutralize, ogni processore cercherà quindi di ottenere un nuovo blocco dal lato sinistro dell'array se il suo blocco di sinistra è stato neutralizzato prima, o dal lato destro se il suo blocco di destra è stato neutralizzato prima e neutralizzerà il nuovo blocco con quello appena consegnato dal Processore P0. Se entrambi i blocchi sono stati neutralizzati, il processore P0 invierà i blocchi da entrambe le estremità. I processori continuano i passaggi precedenti fino a quando non completano tutti i blocchi. Infine avremo che, ogni processore ha al massimo un blocco non neutralizzato e invia la sua copia al processore P0 che si occuperera ad aggiornare l'array iniziale. Una volta aggiornato tutto l'array,i processori inviano a P0 un numero che contiene i blocchi di sinistra che sono stati neutralizzati, LN (Neutralizzato a sinistra) e un numero che contiene blocchi di destra che sono stati neutralizzati, RN (Neutralizzato a destra).
In questo caso ci saranno al massimo MAX_PROCESSORS di blocchi.

4.2 Round 2: partizione sequenziale dei dati

Lo scopo del secondo round è terminare ciò che è iniziato nella partizione parallela. Quando la partizione parallela completa di neutralizzare i blocchi, ci troveremo con parti di blocchi compresi tra [0, LN - 1) e [RN, N - 1), i quali risultano essere ordinati correttamente rispetto al pivot. I restanti blocchi P che possono essere posizionati in qualsiasi posizione sull'array come mostrato nella figura;

Devono essere posizionati al centro dell array, [LN - 1, RN], e se all'interno del intervallo si trovano blocchi gia neutralizzati allora devono essere scambiati con quelli non neuralizzati. Durante questo round il processore P0 prima ordina i P blocchi,e successivamente selezionare un blocco da sinistra e un blocco da destra e utilizza funzione neutralize insieme al pivot precedentemente selezionato, ed esegue la funzione fino a quando tutti i blocchi rimanenti sono esauriti.

Dopo le due fasi di partizione,l'array verrà suddiviso in due array, SplitPoint, i valori minori del pivot si troveranno nel primo array mentre gli altri valori nel secondo array.

4.3 Round 3: partizione del processo

Durante questa fase, l'algoritmo divide tutti processori in due gruppi. Le dimensioni di questi gruppi sono proporzionali alle dimensioni dei sottoarray.

int i_L1 = i_splitPoint; /* Grandezza del blocco di sinistra. */ int i_L2 = i_currentSize - i_L1; /* Grandezza del blocco di destra. */ int i_P2 = round((i_L2 * i_groupSize)/i_currentSize); /* Numero di processi della parte di destra. */

Se la dimensione di un gruppo è zero, allora tutti i processori lavoreranno sull altro array. Una volta distribuiti i processori, il round 3 consiste nell'applicare lo stesso metodo di partizione parallela, del Round Uno al Round tre, in modo ricorsivo. Quando viene assegnato un solo processore a un sottoarray, il processore uscirà dal Round tre.

4.4 Round 4: l'Ordinamento iterativo in parallelo

Durante la fase di ordinamenti iterativo, ogni processore utilizza quicksort per ordinare il sottoarray ottenuto dalla fase tre.

Capitolo 5

Prestazioni

Capitolo 6

Conclusioni

Bibliografia