Progetto di programmazione - Algoritmi Avanzati modulo 2
Anno Accademico 2014/2015

Progetto di Gennaio/Febbraio A.A. 2014/2015

Studente : Andrea Sghedoni
Matricola : 0000736038
Mail : andrea.sghedoni4@studio.unibo.it

INTRODUZIONE

Il progetto prevedeva l'implementazione del calcolo delle somme prefisse di un array, considerando di essere in un'architettura a memoria distribuita.
Il progetto è stato implementato utilizzando il linguaggio C, con l'ausilio delle librerie MPI per la gestione dell'ambiente distribuito, in particolare si è ricorsi all'utilizzo di chiamate primitive per la comunicazione tra diversi processi (siano essi lanciati localmente o distribuiti in una rete).

IMPLEMENTAZIONE

Come prima cosa vengono istanziate diverse variabili che verranno utilizzate all'interno della procedura (nel codice è commentato il significato di ogni singola variabile).
Successivamente il processo master legge, dal file sorgente "in.txt", il numero di elementi che compongono la sequenza in input e tramite la funzione read_from_file, legge tutti gli elementi memorizzandoli in un array.
Quest'ultimo viene allocato, tramite una malloc, soltanto sul master, in quanto l'operazione di lettura dal file dell'intero input spetta esclusivamente ad esso.
Dopodichè viene calcolato il numero di elementi che ogni nodo dovrà processare, supponendo che ogni processo legga dal file solo la prima riga di esso per sapere quanti elementi dovrà considerare. L'implementazione prevede che il numero totale di elementi possa anche non essere multiplo esatto del numero di processi, di conseguenza il resto (remainder) del multiplo verrà processato dal processo master.
ESEMPIO:
Supponiamo che vi siano 100 elementi da scansionare in un'architettura distribuita con 3 processori :

- ogni processo dovrà operare su un array di dimensioni 33, tranne il processo master che dovrà considerare un array di 34 in quanto il resto (100 % 3) vale 1.

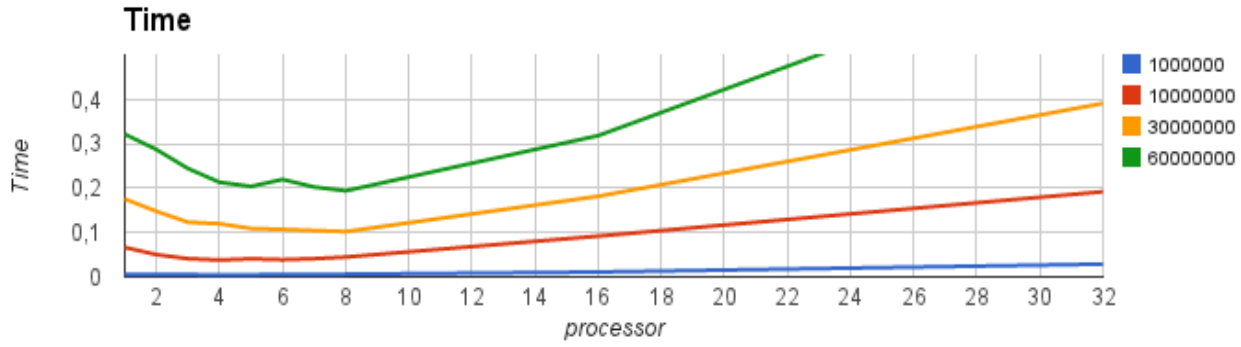
Si potrebbe pensare che vi sia uno sbilanciamento del carico a sfavore del master, nel caso in cui il numero di processi p non sia multiplo della quantità dell'input n, ma ciò può essere considerato ininfluente poichè si prendono in esamine situazioni in cui n>>>>p.
Successivamente i dati di input vengono splittati tra i vari processi tramite l'implementazione di una Scatterv, si è adottata questa soluzione poichè, contrariamente alla Scatter, si possono differenziare le quantità in input per diversi processi (nel caso in cui il processo master debba prendersi carico di elementi in più rispetto agli altri processi, tramite questa chiamata MPI è possibile farlo).
Proseguendo vengono fatte le somme prefisse locali grazie ad un semplice ciclo for e raccolti tutti gli "ultimi elementi" delle somme prefisse di ogni processo nel master, tramite la chiamata MPI Gather.
Il processo master fa la somma prefissa degli "ultimi elementi" ricevuti e inoltra i primi p-1 elementi a tutti gli altri processi, tramite una Scatterv.
Questi sommano l'elemento arrivato in precedenza con le somme prefisse calcolate in precedenza.
Il master ha già la prima porzione dell'array contenente i risultati finali delle somme prefisse, quindi non ha necessità di fare la procedura sopracitata.
Ora le somme prefisse dell'input di partenza sono state calcolate e sono splittate tra i vari processi, di conseguenza come ultimo passo si concatenano tutte nell'array s, del processo master, tramite una chiamata MPI Gather.
Infine viene memorizzato il risultato delle somme prefisse in un file di output "out.txt".
La struttura del sorgente C si basa sullo schema e sulle direttive algoritmiche mostrate nella consegna.

RISULTATI EMPIRICI - SPEEDUP ED EFFICIENZA

Di seguito vengono presentati i risultati ottenuti da varie esperienze, ricavando da essi l'analisi di speedup ed efficienza.
Verranno proposti i grafici di: Tempo di esecuzione (espresso in secondi), Speedup, Efficienza.Il numero di processi preso in considerazione si limita a 1, 2, 3, 4, 5, 6, 7, 8, 16, 32.Il numero di dati in input è stato scelto tra 1000000, 10000000, 30000000, 60000000. Input con una dimensione molto minore tra quelli indicati non erano minimamente significativi per la scalabilità, per il calcolo di speedup ed efficienza; mentre con input maggiori si richiano degli out of memory.

Tempo esecuzione

Tempi misurati sperimentalmente, in secondi:

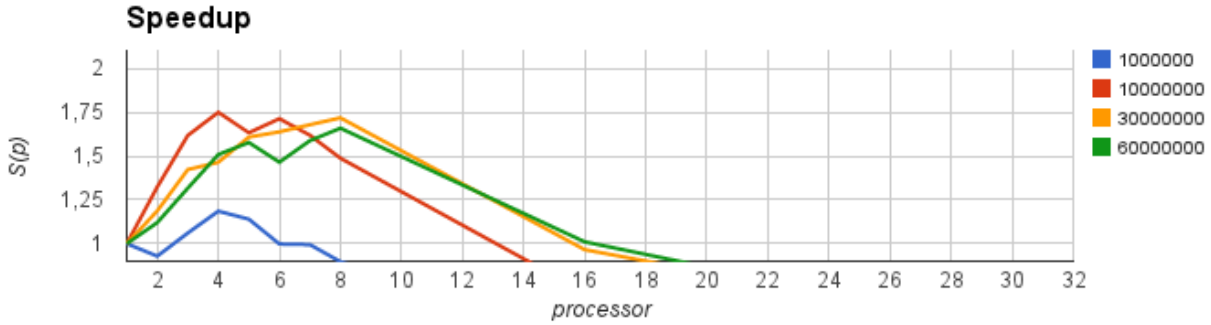


Speedup

Lo speedup è stato calcolato con l'equazione S(p) = T(1)/T(p) dove:

T(1) ← tempo impiegato da un processo a risolvere le somme prefisse

T(p) ← tempo impiegato da p processi a risolvere le somme prefisse

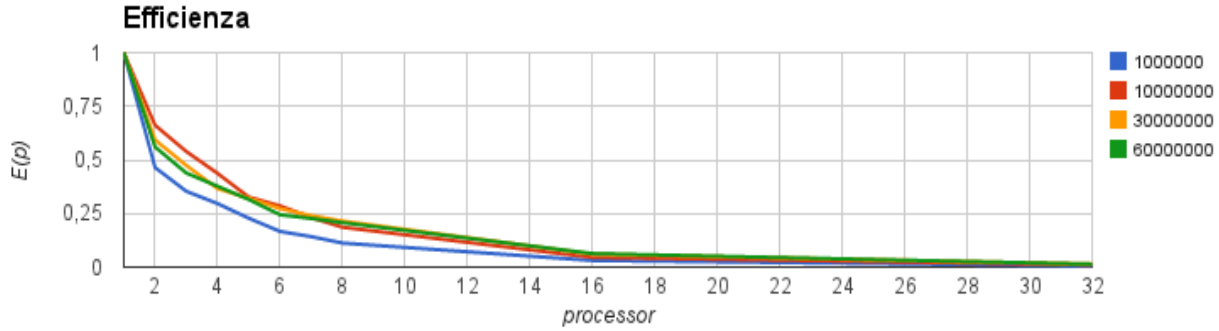


Efficiency

L'efficienza è stata calcolata con la formula E(p) = S(p)/p dove:

S(p) ← Speedup rilevato con p processi

p ← numero di processi



Architettura hardware per il testing e sviluppo:

Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz
N° Processori 8

CONSIDERAZIONI

- **VANTAGGI:**
  L'implementazione adottata è di facile comprensione e svolta in un numero di operazioni consono al problema proposto.
  L'utilizzo delle chiamate MPI collettive è un'altro punto di forza dell'implementazione presentata, in quanto preferibili alle numerose chiamate punto a punto che sarebbe stato necessario attivare.

- **SVANTAGGI**
  La scalabilità, almeno sull'hardware adottato per l'implementazione e per il testing, non è apparsa ottima, anzi per vedere un minimo di riduzione dei tempi di esecuzione è stato necessario ricorrere a dimensioni di input notevoli.
  Si è giunti alla conclusione, anche sulla base dei risultati ottenuti empiricamente (mostrati sopra), che l'onerosità delle primitive di comunicazione MPI superino, in proporzione, il tempo computazionale necessario per il calcolo locale delle somme prefisse.
  Numerose prove effettuate evidenziano che le comunicazioni più onerose sono la prima Scatterv, dove vengono splittati i dati tra i diversi processori, e l'ultima Gather, dove vengono raccolti i risultati nell'array finale del processo master.

Di seguito viene mostrata la tabella dei tempi di esecuzione rilevati, da cui sono stati effettuati i calcoli relativi ai grafici mostrati precedentemente.
Il tempo viene percepito tramite la funzione MPI_Wtime(), non tenendo conto della prima fase di input dal file in.txt e dell'ultima fase per la scrittura su file out.txt del risultato:

T esecution (s)	input			
processors	1000000	10000000	30000000	60000000
1	0,005411	0,066404	0,17539	0,321589
2	0,005826	0,050177	0,147792	0,287094
3	0,005102	0,041054	0,123303	0,244304

4	0,004565	0,03797	0,119792	0,213217
5	0,004748	0,040679	0,109085	0,203941
6	0,005421	0,038772	0,107102	0,219568
7	0,005441	0,04103	0,104503	0,202571
8	0,006032	0,04465	0,10212	0,193912
16	0,010976	0,091866	0,181634	0,318281
32	0,028793	0,191935	0,391002	0,735119