Di Riccio Tommaso 615257 29/05/2024

Compilazione ed esecuzione

Per compilare è possibile utilizzare il comando:

mpicxx -std=c++20 -Wall -fopenmp -O3 -o nkeyspar nkeyspar.cpp (or use makefile)

Per l'esecuzione il comando è:

srun --mpi=pmix --nodes=[#nodes] -n [#proc] ./nkeyspar [nkeys] [length] (or use SLURM script)

dove

- *nkeys* è il numero di chiavi utilizzare nell'esecuzione
- *length* è la lunghezza dello stream, cioè il numero di coppie di chiavi randomiche generate

Problemi affrontati e soluzioni

Nella versione parallela MPI i task "compute", cioè i prodotti tra matrici, sono distribuiti tra i processi disponibili. Il processo Master (rank 0) genera le coppie di chiavi randomiche e, quando richiesto, invia i dati necessari per eseguire il metodo compute(...) ai processi, selezionandoli in modalità Round Robin.

Quando una chiave viene generata in iterazioni diverse è necessario che la precedente computazione sulla chiave sia stata completata e che il valore all'interno della map sia aggiornato. Per garantire ciò il processo master mantiene una map "*pending*", in cui è memorizzato se un risultato è ancora da ricevere per una certa chiave. Dunque, dopo la generazione, viene controllato se per le chiavi sono attesi dei risultati. In caso affermativo, il processo Master riceve risultati pronti dai processi, fino a quando non ottiene il valore calcolato per la coppia di chiavi generate. Aggiorna poi il valore all'interno della map e prosegue la computazione inviando nuove richieste di calcolo ai processi. La scelta di ricevere, oltre al risultato per la chiave desiderata, possibilmente risultati per altre chiavi permette di evitare uno stallo, senza compromettere le prestazioni.

Nella seconda parte ("compute last values") è possibile che la chiave sia utilizzata parallelamente in più *compute*, in quanto non sono presenti dipendenze da tenere in considerazione. Le richieste sono inviate con una Send bloccante perché in questa fase il processo Master non ha altro lavoro da eseguire che distribuire i calcoli tra i processi e attendere i risultati. È stata testata anche una versione in cui lo stesso processo master partecipa a parte dei calcoli da eseguire. Questa versione migliorava leggermente le prestazioni con un numero di processori basso (<4) ma peggiorava notevolmente le prestazioni con un alto numero di processori ed è stata dunque scartata.

Dall'altra parte, i processi utilizzati per i prodotti tra matrice ricevono i dati (4 long: c1, c2, key1, key2) dal processo Master e chiamano il metodo *compute(...)*.

Per inviare il risultato al processo Master viene utilizzato uno structure datatype RESULT_T che consiste in una coppia (long, float) in cui il primo elemento rappresenta la chiave e il secondo elemento il risultato del calcolo.

Nelle comunicazioni MPI il tag 1 viene utilizzato per inviare le richieste di calcolo, il tag 2 per le risposte e il tag 9 per indicare la terminazione del programma.

Esperimenti e risultati

Nella seguente tabella sono riportati i tempi, su una media di 5 iterazioni, per la **versione sequenziale** utilizzando il parametro *nkeys* con valore pari a 2, 100 e 1000 e il parametro *length* fissato a 100000

Sequenziale			
Nkeys	2	100	10000
Length	1000000	1000000	1000000
Tempo (sec)	11,64	6,93	119,86

Gli stessi parametri sono stati utilizzati per eseguire la **versione parallela** utilizzando 2, 3, 4, 8, 16 e 24 processi. Per i seguenti risultati sono stati utilizzati 2 nodi del cluster.

2 Processi - 2 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	17,24	9,27	158,46

3 Processi - 2 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	9,31	6,24	79,79

4 Processi - 2 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	9,76	5,79	55,37

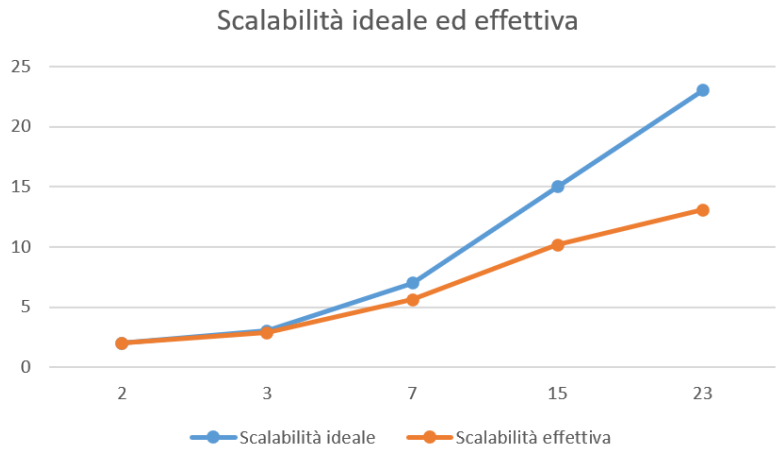
8 Processi - 2 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	8,87	5,07	28,24

16 Processi - 2 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	8,55	4,75	15,61

24 Processi - 2 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	8,23	4,64	12,14

Speedup		Nkeys		
		2	100	1000
Processi	2	0,68	0,75	0,76
	3	1,25	1,11	1,50
	4	1,19	1,20	2,16
	8	1,31	1,37	4,24
	16	1,36	1,46	7,68
	24	1,41	1,49	9,87

Analizzando la scalabilità per Nkeys=1000 si ottiene:



#Processi slave	Tempo	Scalabilità
1	158,46	-
2	79,79	1,99
3	55,37	2,86
7	28,24	5,61
15	15,61	10,15
23	12,14	13,05

Utilizzando invece fino ad un massimo di 8 nodi, dividendo in modo equo i processi, si ottiene:

2 Processi - 2 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	17,24	9,27	158,46

3 Processi - 3 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	9,43	6,78	79,99

4 Processi - 4 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	9,49	6,15	54,09

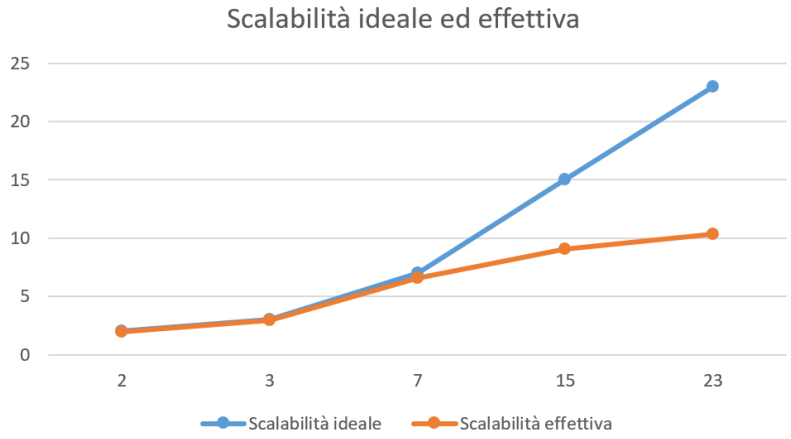
8 Processi - 8 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	10,24	5,96	24,16

16 Processi - 8 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	10,16	6,06	17,45

24 Processi - 8 Nodi			
Nkeys	2	100	1000
Length	1000000	1000000	1000000
Tempo (sec)	9,87	5,56	15,36

Speedup		Nkeys		
		2	100	1000
Processi	2	0,68	0,75	0,76
	3	1,23	1,02	1,50
	4	1,23	1,13	2,22
	8	1,14	1,16	4,96
	16	1,15	1,14	6,87
	24	1,18	1,25	7,80

Analizzando la scalabilità per Nkeys=1000 si ottiene:



#Processi slave	Tempo	Scalabilità
1	158,46	-
2	79,99	1,98
3	54,09	2,93
7	24,16	6,56
15	17,45	9,08
23	15,36	10,32

Quando *nkeys*=2, con più 3 processi si ottengono performance poco migliori o peggiori. Questo è ragionevole in quanto, non considerando il processo master, i restanti due processi si occupano delle uniche due chiavi che vengono generate. Utilizzando più processi è difficile sfruttare le risorse a causa delle dipendenze tra le iterazioni.

Utilizzando un maggior numero di chiavi, *nkeys*=100, si ha un miglioramento nei tempi di esecuzione fino a 24 processi sia con 2 nodi che con 8. Nonostante ciò, lo speedup è fortemente sublineare.

Incrementando ulteriormente il range di chiavi generabili, *nkeys*=1000 si nota come nuovamente le miglior performance è ottenuta con 24 processi, tuttavia, in questo caso, gli speedup sono decisamente più alti. Con questa configurazione di parametri è possibile anche notare una buona scalabilità fino a 8 processi, non presente con valori di *nkeys* più bassi.
Considerando solo il numero di processi slave (i processi che calcolano il prodotto tra matrici) si può notare come la scalabilità [$T_{par}(1)/T_{par}(p)$] cresca in modo praticamente lineare fino a 7 processi, specialmente utilizzando 2 nodi.