Corso di Algoritmi Avanzati—modulo 2

Primo progetto di programmazione—Anno Accademico 2014/2015

Moreno Marzolla

Versione 1.0 del 10/12/2014

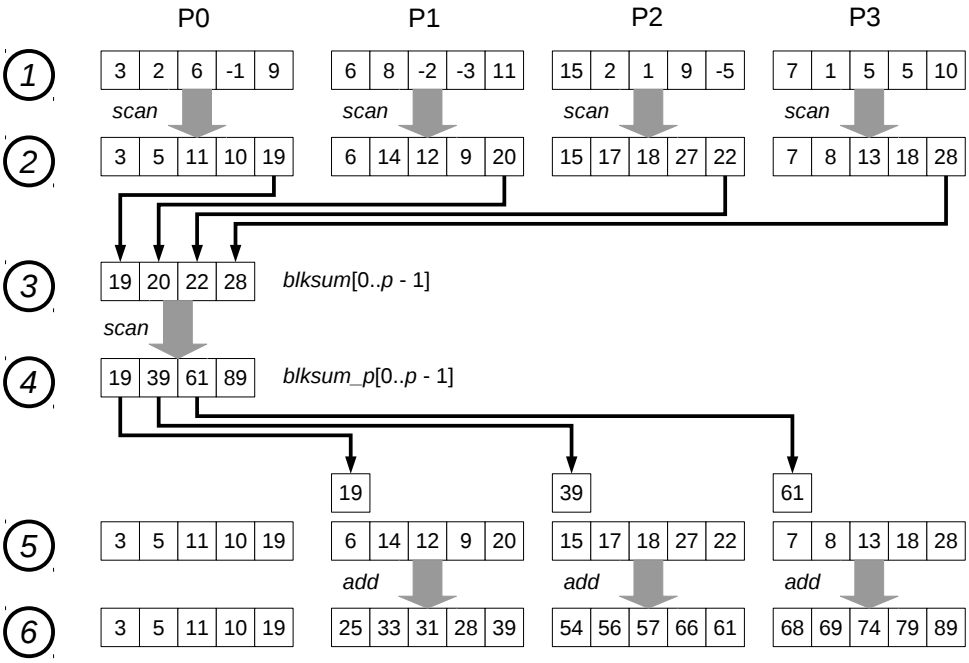
Prima versione di questo documento

Descrizione del progetto

Scopo del progetto è l'implementazione di un algoritmo parallelo per il calcolo delle somme prefisse su architetture a memoria distribuita utilizzando MPI. Si noti che la funzione `MPI_Scan` calcola le somme prefisse di P elementi, ciascuno presente in uno dei P processi MPI che eseguono la funzione `MPI_Scan`. In questo progetto verrà richiesta l'implementazione di un algoritmo generico in grado di calcolare le somme prefisse di un array di n elementi, con n arbitrario.

Sia $a[0..n-1]$ un array di float. Scopo del programma è di calcolare un secondo array di float $s[0..n-1]$, tale che per ogni $i = 0, \ldots n-1$ si abbia $s[i] = a[0] + \ldots + a[i]$. Quindi $s[0] = a[0]$, $s[1] = a[0] + a[1]$, $s[2] = a[0] + a[1] + a[2]$, e così via. L'array $s[0..n-1]$ contiene le somme prefisse di a, come verrebbero calcolate dall'operazione di *scan inclusivo* vista a lezione.

Il programma deve operare seguendo i passi seguenti (si faccia riferimento alla figura).



1. Il processo master partiziona l'array $a[0..n-1]$ in P blocchi di n/P elementi ciascuno; ogni blocco viene assegnato ad uno dei P processi.
2. Ciascun processo calcola le somme prefisse del proprio blocco, utilizzando l'algoritmo sequenziale.
3. Ciascun processo comunica al master l'ultima somma prefissa del proprio blocco. Sia *blksum*$[0..P-1]$ l'array composto da tutte le ultime somme prefisse ricevute dal master.
4. Il master calcola le somme prefisse di *blksum*$[0..P-1]$, utilizzando l'algoritmo sequenziale. Chiamiamo *blksum_p*$[0..P-1]$ l'array delle somme prefisse.

5. Il master distribuisce il valore di *blksum_p*[*i*] al processo *i* + 1, per ogni *i* = 0, ... *P* - 2.
6. Ciascun processo somma il valore ricevuto al passo precedente a tutti i valori delle somme prefisse precedentemente calcolate.
7. Ciascun processo comunica il proprio blocco al master, che li concatena ottenendo l'array delle somme prefisse *s*[0..*n* - 1] di *a*[0..*n* - 1].

Il master deve leggere i valori di *a*[0..n - 1] da un file di testo `in.txt` che si trova nella stessa directory dell'eseguibile. Il file `in.txt` è composto da di *n* + 1 righe di testo: la prima riga contiene l'intero *n*, e le successive *n* righe contengono i valori degli elementi *a*[0..*n* - 1]. Si può far uso della funzione `fscanf()` per la lettura del file. Al termine dell'esecuzione, il master scriverà i valori delle somme prefisse in un file `out.txt`, avente il medesimo formato: la prima riga contiene l'intero *n*, e le successive contengono i valori di *s*[0..*n* - 1].

Si noti che i passi di cui sopra devono essere intesi come uno schema di massima; è parte del progetto la scelta delle primitive MPI da utilizzare, e il layout più opportuno dei dati in memoria (ad esempio, allocando elementi in più o in meno nei vari vettori, in base alle esigenze, riutilizzare gli array, usare nomi di variabili diverse...). Nella relazione è possibile motivare le scelte implementative, indicando i pro e contro dell'implementazione consegnata e le eventuali limitazioni.

Modalità di svolgimento del progetto

- Il progetto deve essere svolto individualmente. Non è consentito condividere codice o relazione con altri studenti, né utilizzare software disponibile in rete.
- Il progetto deve essere realizzato in un singolo file sorgente chiamato `scan.c`. Il sorgente deve essere adeguatamente commentato. All'inizio del file sorgente deve essere incluso un commento che riporta cognome, nome e numero di matricola dell'autore/autrice, nonché il comando da usare per la compilazione (vedi punto successivo).
- Il progetto deve essere implementato in linguaggio C come applicazione a riga di comando, facendo uso delle librerie MPI. Il progetto deve essere compilabile ed eseguibile senza errori su una delle macchine Linux dei laboratori studenti (sistema operativo Debian GNU/Linux 7.7), oppure sull'immagine dei laboratori virtuali (http://www.virtlab.unibo.it/index.html) mediante il comando

  ```
  mpicc -Wall scan.c -o scan [eventuali altri flag]
  ```

  Indicare nel commento iniziale del sorgente e/o nella relazione la riga di comando corretta per la compilazione nel caso siano richiesti altri flag, ad esempio nel caso in cui sia necessario includere librerie di sistema. Nota: sulle macchine del laboratorio è installato OpenMPI (pacchetti `openmpi-bin`, `libopenmpi-dev` ed `openmpi-doc` per la documentazione).
- Per verificarne la correttezza, il programma verrà eseguito mediante il comando

  ```
  mpirun -n P ./scan
  ```

  dove *P* indica il numero di processi MPI da creare. Tutte le istanze saranno lanciate localmente, in modo da evitare ogni problema con la rete. Sarà sempre disponibile un file `in.txt` avente il formato corretto.
- Si può assumere che la dimensione *n* dell'array a sia sempre (molto) maggiore del numero di processi *P*, e che gli array *a*[] ed *s*[] possano sempre essere interamente contenuti nella memoria del processo master, insieme a tutti gli eventuali altri array e/o variabili

temporanee.

- Assieme al sorgente deve essere consegnare una relazione in formato pdf in un file chiamato `relazione.pdf`. La relazione non deve superare la lunghezza di cinque facciate in formato A4 (font non inferiore a 10 punti), e deve descrivere ad alto livello l'implementazione con particolare riguardo alla strategia di parallelizzazione adottata e gli eventuali limiti e/o potenzialità di tale strategia. Indicare il proprio cognome, nome e numero di matricola all'inizio della relazione. La relazione deve includere una sintetica analisi sperimentale della scalabilità dell'algoritmo realizzato, mostrando i grafici di speedup ed efficienza. Non è richiesto che l'analisi di scalabilità venga effettuata sulle macchine del laboratorio. È possibile effettuare i test lanciando tutte le istanze localmente (anche in numero superiore al numero di core disponibili); ovviamente non terrò conto dei tempi di esecuzione in termini assoluti, ma solo del fatto che l'analisi di speedup ed efficienza sia svolta correttamente.

Scadenza e modalità di consegna

Sono previste due scadenze per la consegna:

1. entro le **ore 12:00** (mezzogiorno) di **lunedì 19 gennaio 2015**, oppure
2. entro le **ore 12:00** (mezzogiorno) di **lunedì 9 febbraio 2015**

Ricordo che è necessario ottenere una valutazione positiva del progetto per poter partecipare alla prova scritta del modulo 1. La valutazione verrà comunicata via mail, e se positiva resterà valida per l'intero anno accademico 2014/2015, ossia fino all'appello di esami di gennaio/febbraio 2016 (compreso).

La consegna deve avvenire inviando una mail dal proprio indirizzo istituzionale @studio.unibo.it a moreno.marzolla@unibo.it con subject

Consegna Progetto Algoritmi Avanzati 2014/2015

Nella mail vanno indicati cognome, nome, numero di matricola del mittente; siete pregati di indicare se intendete sostenere l'esame del modulo 1 nell'appello immediatamente successivo alla data di consegna del progetto, in modo da ricevere priorità nella correzione. Alla mail deve essere allegato un archivio in formato `.zip` oppure `.tar.gz` contenente il sorgente e la relazione in formato pdf; chi lo desidera può includere altri files, ad esempio un `Makefile` per la compilazione (non obbligatorio). L'allegato sarà denominato con il cognome dell'autore (es., `Rossi.zip` oppure `Rossi.tar.gz`), e conterrà una directory con lo stesso nome (es., `Rossi/`) contenente a sua volta il sorgente e la relazione in formato pdf.

Nota per gli utenti Apple: gli archivi .zip prodotti sotto MacOSX spesso includono una directory `.DS_Store/`. Generate l'archivio da riga di comando in modo da non includerla.

Valutazione dei progetti

Per ottenere una valutazione positiva è necessario che il programma compili sulle macchine Linux del laboratorio studenti e funzioni correttamente. Ulteriori elementi di cui si terrà conto:

- Efficienza della strategia di parallelizzazione adottata;
- Generalità della soluzione implementata; ad esempio, il programma fornito dovrebbe poter funzionare con array di dimensioni qualsiasi (maggiori del numero di processori P). Verranno comunque accettate soluzioni che impongono vincoli (ad esempio, che richiedono che n sia multiplo di P), con conseguenti penalizzazioni in sede di valutazione.
- Qualità della relazione, in termini di chiarezza, correttezza e presentazione del contenuto;
- Qualità del codice, in termini di semplicità e chiarezza. Verrà penalizzato il ricorso a micro-ottimizzazioni inutili, nonché codice contorto o ridondante.