Risoluzione di un sistema lineare a freccia

Consideriamo il problema della risoluzione di un sistema lineare a freccia (detto anche bordato a blocchi, o block bordered). È molto frequente incontrare questo tipo di sistemi lineari nella risoluzione di problemi reali composti da blocchi interconnessi tra loro, come la progettazione di circuiti integrati VLSI o l'ottimizzazione strutturale. Vedremo come è possibile sfruttare le particolarità di questo problema per sviluppare un metodo di risoluzione parallelo. In generale, un sistema a freccia ha la seguente forma:

$$\begin{pmatrix} A_0 & & & & B_0 \\ & A_1 & & & B_1 \\ & & \ddots & & \vdots \\ & & & A_{N-1} & B_{N-1} \\ C_0 & C_1 & \cdots & C_{N-1} & A_s \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{N-1} \\ \boldsymbol{x}_s \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_{N-1} \\ \boldsymbol{b}_s \end{pmatrix} \qquad det(A_i) \neq 0$$

Ponendo:

$$A_I = diag(A_0, \ldots, A_{N-1})$$

$$B = \begin{pmatrix} B_0 \\ \vdots \\ B_{N-1} \end{pmatrix}$$

$$C = (C_0, \ldots, C_{N-1})$$

$$\boldsymbol{x}_I = \begin{pmatrix} \boldsymbol{x}_0 \\ \vdots \\ \boldsymbol{x}_{N-1} \end{pmatrix}$$

$$\boldsymbol{b}_I = \begin{pmatrix} \boldsymbol{b}_0 \\ \vdots \\ \boldsymbol{b}_{N-1} \end{pmatrix}$$

il sistema può essere riscritto come:

$$A_I \boldsymbol{x}_I + B\boldsymbol{x}_s = \boldsymbol{b}_I \qquad (1)$$

$$C_I \boldsymbol{x}_I + A_s\boldsymbol{x}_s = \boldsymbol{b}_s \qquad (2)$$

Si noti come la matrice A_I sia fortemente sparsa, in quanto ha tutti gli elementi nulli eccetto i blocchi sulla diagonale. Quando si sviluppa un algoritmo parallelo, il primo passo è quello di identificare se ci sono operazioni indipendenti che possono essere distribuite tra i vari processori. Apparentemente, in questo sistema non ci sono operazioni di questo tipo e quindi, a prima vista, si potrebbe essere portati a risolvere il sistema utilizzando la decomposizione LU parallela oppure un qualunque metodo iterativo. Tuttavia è possibile riscrivere il problema tramite una trasformazione in modo da evidenziare sottoproblemi indipendenti.

Moltiplicando la (1) per CA_I^{-1} e sottraendo il risultato dalla (2) si ha:

$$(A_s - CA_I^{-1}B)\boldsymbol{x}_s = \boldsymbol{b}_s - CA_I^{-1}\boldsymbol{b}_I$$

ovvero

$$\hat{A}\boldsymbol{x}_s = \hat{\boldsymbol{b}}$$

dove

$$\hat{A} = A_s - (C_0, \ldots, C_{N-1}) \begin{pmatrix} A_0^{-1} & & \\ & \ddots & \\ & & A_{N-1}^{-1} \end{pmatrix} \begin{pmatrix} B_0 \\ \vdots \\ B_{N-1} \end{pmatrix} = A_s - \sum_{i=0}^{N-1} C_i A_i^{-1} B_i$$

e

$$\hat{\boldsymbol{b}} = \boldsymbol{b}_s - (C_0, \ldots, C_{N-1}) \begin{pmatrix} A_0^{-1} & & \\ & \ddots & \\ & & A_{N-1}^{-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{b}_0 \\ \vdots \\ \boldsymbol{b}_{N-1} \end{pmatrix} = \boldsymbol{b}_s - \sum_{i=0}^{N-1} C_i A_i^{-1} \boldsymbol{b}_i$$

In questo modo è stato possibile determinare le componenti incognite $\boldsymbol{x}_s$. Ora, osservando che l'equazione (1) è un sistema diagonale a blocchi, è possibile sfruttare $\boldsymbol{x}_s$ per calcolare la soluzione risolvendo N sistemi indipendenti.

È possibile schematizzare l'algoritmo di risoluzione coi seguenti passi:

1. Calcolare la fattorizzazione LU delle matrici $A_0, \ldots, A_{N-1}$;
2. Calcolare $\hat{A} = A_s - \sum_{i=0}^{N-1} C_i A_i^{-1} B_i$ e $\hat{\boldsymbol{b}} = \boldsymbol{b}_s - \sum_{i=0}^{N-1} C_i A_i^{-1} \boldsymbol{b}_i$;
3. Risolvere il sistema $\hat{A}\boldsymbol{x}_s = \hat{\boldsymbol{b}}$ per determinare $\boldsymbol{x}_s$;
4. Calcolare le restanti componenti del vettore soluzione $\boldsymbol{x}$ risolvendo N sistemi lineari $A_i \boldsymbol{x}_i = \boldsymbol{b}_i - B_i \boldsymbol{x}_s, \quad i = 0, \ldots, N-1$.

Osserviamo che il punto 1 è intrinsecamente parallelo, in quanto devono essere risolti i N problemi indipendenti di fattorizzazione delle matrici A_i.
Il punto 2 può essere eseguito in parallelo solo parzialmente; infatti ogni processore può eseguire i prodotti $C_i A_i^{-1} B_i$ e $C_i A_i^{-1} \boldsymbol{b}_i$, ma dovrà essere eseguita una comunicazione per calcolare la somma, dato che occorre raccogliere in un unico processore tutti i prodotti parziali per sommarli.
Il punto 3 non è intrinsecamente parallelo. Può essere risolto in sequenziale dal processore master o in alternativa può essere risolto con un algoritmo LU parallelo.
Una volta determinato $\boldsymbol{x}_s$, è possibile eseguire il punto 4 in parallelo; infatti i N sistemi da risolvere sono indipendenti e la loro risoluzione può essere intrinsecamente distribuita tra i processori disponibili.

Osserviamo infine che nella pratica, le matrici A_i non verranno mai fisicamente invertite. Infatti, lo scopo è quello di calcolare i prodotti $A_i^{-1} B_i$ e $A_i^{-1} \boldsymbol{b}_i$; notiamo che calcolare $A_i^{-1} \boldsymbol{b}_i$ corrisponde a determinare la soluzione del sistema lineare $A_i \boldsymbol{x} = \boldsymbol{b}_i$. Analogamente, se si rappresenta la matrice B_i come l'insieme delle sue colonne $[B_{i0} B_{i1} \cdots B_{ik-1}]$, dove B_{ik} è la k-esima colonna della matrice B_i, è possibile calcolare le colonne del prodotto $A_i^{-1} B_i$ risolvendo i k sistemi lineari $A_i \boldsymbol{x}_j = B_{ij}$ per $j = 0, \cdots, k-1$. Si noti che è sufficiente calcolare la decomposizione LU di A_i inizialmente e riutilizzarla per risolvere ogni sistema lineare.

Si può schematizzare l'algoritmo risolutivo parallelo nel seguente modo:

Distribuzione dei dati: Per semplicità si suppone che il numero di processori coincida con un sottomultiplo di N. Ogni processore i genera i seguenti elementi del problema: A_i, B_i, C_i. Si suppone una distribuzione per blocchi contigui: per esempio, con $N = 4$ e numero di processori $P = 2$, i blocchi A_i, B_i, C_i con $i = 0, 1$ vanno al processore 0, quelli con $i = 2, 3$ al processore 1. Il processore 0 genera i vettore di elementi noti b, inizializzando ogni elemento a 1, successivamente ne distribuisce i blocchi seguendo la stessa distribuzione utilizzata per le matrici. Il processore 0 memorizza anche A_s e b_s.

1. Ogni processore esegue la decomposizione LU delle proprie matrici A_i generando le due matrici L_i e U_i;
2. Ogni processore calcola il termine $\boldsymbol{d}_i = C_i A_i^{-1} \boldsymbol{b}_i$ per ogni matrice A_i ad esso destinata, sfruttando la decomposizione LU calcolata al punto 1, successivamente ne esegue la somma;
3. Tramite una comunicazione di tipo Reduce con somma sui termini $\boldsymbol{d}_i$, il processore master ottiene la somma $\sum_{i=0}^{N-1} C_i A_i^{-1} \boldsymbol{b}_i$ e con questa calcola $\hat{\boldsymbol{b}}$;
4. Ogni processore calcola il termine $D_i = C_i A_i^{-1} B_i$ mediante la risoluzione degli n sistemi lineari $A_i \boldsymbol{x}_j = B_{ij}$;
5. Tramite una comunicazione di tipo Reduce con somma sui termini D_i, il processore master ottiene la somma $\sum_{i=0}^{N-1} C_i A_i^{-1} B_i$ e con questa calcola $\hat{A}$;
6. Il processore master risolve il sistema lineare $\hat{A}\boldsymbol{x}_s = \hat{\boldsymbol{b}}$ per determinare $\boldsymbol{x}_s$;
7. La soluzione $\boldsymbol{x}_s$ viene viene comunicato a tutti i processori tramite una Broadcast;
8. Ogni processore j-esimo risolve i sistemi lineari $A_i \boldsymbol{x}_i = \boldsymbol{b}_i - B_i \boldsymbol{x}_s$, con i ad esso dedicati.
9. Le componenti $\boldsymbol{x}_i$ vengono riunite in un unico vettore tramite una comunicazione di tipo Gather. Unite assieme a $\boldsymbol{x}_s$, formano la soluzione al problema iniziale.

Suggerimenti:
Si considerino blocchi di dimensione BS.
Sfruttare le funzioni presenti nel file sorgente `res.c`. In particolare, per generare i blocchi A_i, utilizzare la funzione `genera_Ai`, passando il puntatore al blocco da riempire, il numero di righe (e di colonne), il coefficiente $i+1$:

```
genera_Ai( A, BS, i+1)
```

Il blocco A_s dovrà essere generato sempre con la funzione `genera_Ai`, e con coefficiente $N+1$. Utilizzare la funzione `genera_Bi` per generare sia i blocchi B_i che C_i. Per il calcolo della fattorizzazione LU, utilizzare la funzione:

```
void LU(double *A,int n)
```

già implementata nell'esercitazione precedente: viene riportata per semplicità nel file `res.c`.
Per la risoluzione dei sistemi, utilizzare le funzioni:

```
void sost_indietro(double *U, int m, double * B, int s, double *X)
void elim_avanti(double *L, int m, double *B, int s, double *X)
```

che permettono di calcolare s sistemi con una singola chiamata.
Infine, per il prodotto matrice-matrice, utilizzare la funzione `my_gemm` che implementa il prodotto matrice matrice. Fornire prima una implementazione seriale, poi una implementazione parallela utilizzando MPI. Per i grafici di speedup, efficienza e funzione di Kuck considerare $BS = 256$, $N = 128$ e $P = 2, 4, 8, 16$.