UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Computació Paral·lela (CPA)

Pràctica 4: Computació Matricial amb MPI

Curs 2017/18

Índex

Introducció

Aquesta última pràctica se centra en la implementació en paral·lel mitjançant MPI d'un problema numèric més avançat, concretament la resolució de sistemes d'equacions lineals. L'objectiu de la pràctica és enfrontar-se a un problema de major envergadura i amb més possibilitats de millora.

El material de partida per a realitzar la pràctica consisteix en el fitxer `sistbf.c`, que implementa la versió paral·lela per blocs de files. Haurà de modificar-se perquè contemple una distribució cíclica per files.

Aquesta pràctica es compon de 2 sessions:

- Sessió 1: Estudi del problema i implementació de la versió paral·lela cíclica.
- Sessió 2: Millora de la versió paral·lela cíclica i mesura experimental de les prestacions.

1 Resolució de sistemes d'equacions lineals

Volem resoldre un sistema d'equacions lineals, descrit en notació matricial com $Ax = b$, on A és una matriu, b és el vector "part dreta" (o de termes independents), i x és el vector solució. Ens centrem en el cas en què A és quadrada, i denotem per n la seua dimensió:

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix}, \quad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}, \quad b = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}. \tag{1}$$

Así, por ejemplo, la primera ecuación sería

$$a_{0,0}x_0 + a_{1,0}x_1 + \cdots + a_{n-1,0}x_{n-1} = b_0. \tag{2}$$

Descomposició LU

```
U = A;
Per a k = 0, ..., n-2
  si ukk = 0 aleshores abandona
  lkk = 1
  Per a i = k+1, ..., n-1
    m = uik/ukk
    lik = m
    uik = 0
    Per a j = k+1, ..., n-1
      uij = uij - m*ukj
    Fper
  Fper
Fper
lnn = 1
```

Eliminació Progressiva

```
% Ly = b
y = b
Per a i = 0, 1, ..., n-1
  Per a j = i+1, ..., n-1
    yj = yj - lji*yi
  Fper
Fper
```

Substitució Regressiva

```
% Ux = b
x = b
Per a i = n-1, ..., 0
  xi = xi/uii
  Per a j = i-1, ..., 0
    xj = xj - uji*xi
  Fper
Fper
```

Figura 1: Algorismes de descomposició LU i resolució de sistemes triangulars.

Així, per exemple, la primera equació seria

$$a_{1,1}x_1 + a_{2,1}x_2 + \cdots + a_{n,1}x_n = b_1. \tag{3}$$

Es vol calcular el vector x que satisfà les n equacions simultàniament. El sistema té solució única (és compatible i determinat) quan la matriu A té un determinant diferent de zero. Existeixen diverses tècniques directes i iteratives per a la resolució d'un sistema d'equacions, destacant entre les tècniques directes la factorització LU.

La factorització LU consisteix en l'obtenció d'un parell de matrius, una triangular inferior unitat (tots els elements per sobre de la diagonal principal a zero i la pròpia diagonal principal a un) i una altra triangular superior (tots els elements per sota de la diagonal principal a zero i sense restriccions sobre la diagonal principal), ambdues de la mateixa dimensió que la matriu de partida, tals que el seu producte és igual a A.

D'aquesta forma, la resolució del sistema d'equacions equival a la resolució de dos sistemes triangulars:

$$\left.\begin{array}{c} A = LU \\ Ax = b \end{array}\right\} \quad \longrightarrow \quad LUx = b \quad \longrightarrow \quad \left\{\begin{array}{c} Ly = b \\ Ux = y \end{array}\right. . \tag{4}$$

El càlcul de la descomposició LU es pot realitzar mitjançant l'eliminació Gaussiana, utilitzant els elements diagonals de la matriu com a pivots i fent zeros per sota de la diagonal en cada columna.

Els sistemes triangulars poden resoldre's aplicant els algorismes d'eliminació progressiva (per al cas de la matriu triangular inferior unitat) i substitució regressiva (per al cas de la matriu triangular superior).

1.1 Algorisme seqüencial

Els algorismes necessaris per a resoldre el problema es mostren en la Figura 1. Cadascun dels algorismes aborda una de les fases de la resolució: factorització LU, resolució del sistema triangular inferior i resolució del sistema triangular superior.

L'algorisme LU realitza $n-1$ etapes (bucle k), en cadascuna de les quals es fan zeros per sota de la diagonal en la columna k-èsima. Aquesta columna té $n-k-1$ elements que cal anul·lar (bucle i). En cada iteració i es modifica la fila i completa, però atès que els primers k elements són ja zero, el bucle j només recorre els elements a partir del $k+1$. L'actualització de la fila i involucra la fila k (denominada fila pivot) i utilitza un factor m denominat multiplicador, que és igual a l'element que s'està anul·lant (a_{ik}) dividit per l'element diagonal de la fila pivot (a_{kk}), també anomenat element pivot.

Per a facilitar la comprovació dels algorismes, així com l'anàlisi de les prestacions, s'utilitzaran unes matrius especials denominades de Toeplitz, que es caracteritzen per tenir totes les diagonals constants, és a dir, els elements de la diagonal principal són tots iguals, així com en el cas de les altres sub-diagonals i

$$\begin{bmatrix} c_0 & f_1 & f_2 & f_3 & f_4 \\ c_1 & c_0 & f_1 & f_2 & f_3 \\ c_2 & c_1 & c_0 & f_1 & f_2 \\ c_3 & c_2 & c_1 & c_0 & f_1 \\ c_4 & c_3 & c_2 & c_1 & c_0 \end{bmatrix}$$

```
function [A, b] = creatoep(c,f)
  n = length(c);
  for i=0:n-1
      A(i,i) = c(0);
      b(i) = c(0);
      for j=0:i-1
          A(i,j) = c(i-j);
          A(j,i) = f(i-j);
          b(i) = b(i) + A(i,j);
          b(j) = b(j) + A(j,i);
      end
  end
```

Figura 2: Exemple d'una matriu Toeplitz i algorisme per a la seua creació a partir dels vectors de fila (`f`) i columna (`c`).

super-diagonals. Una matriu de Toeplitz es construeix a partir de dos vectors (f i c) que defineixen els valors de la primera fila i columna (el primer valor de tots dos vectors ha de ser el mateix). A més assegurarem que la matriu és definida positiva incrementant sensiblement el valor de la diagonal. L'algorisme mostrat en la Figura 2 genera una matriu de Toeplitz a partir de f i c, i a més genera un vector b tal que la solució del sistema $Ax = b$ és $x = [1, 1, \ldots, 1]^T$, amb el que és molt senzill comprovar que l'algorisme està funcionant correctament.

L'algorisme de descomposició LU s'ha plantejat de manera que l'eixida sobreescriga a l'entrada. Així, la matriu A, després de l'execució de la funció que realitza la factorització, conté les matrius L i O. De la mateixa forma, en la resolució dels sistemes triangulars, el vector part dreta se sobreescriu amb la solució del sistema d'equacions. En la Figura 3 es mostren aquests algorismes.

2 Implementació paral·lela

La implementació paral·lela de la resolució d'un sistema d'equacions lineals pretén realitzar de forma concurrent diverses operacions independents.

El paral·lelisme més evident succeeix per files, considerant que l'actualització de cada fila és independent (bucle i). D'aquesta forma, es podria repartir la matriu A per files i realitzar l'actualització en paral·lel. Aquest procés suposaria enviar la fila pivot a tots els processos involucrats abans de començar l'actualització de les seues files. La Figura 4 mostra un esquema de com els bucles j s'executen en paral·lel.

Per a resoldre els sistemes triangulars, el vector b es troba replicat en tots els processos i van actualitzant-se en paral·lel cadascun dels seus elements a partir de les noves solucions obtingudes. En el cas d'una implementació per files existeixen múltiples distribucions de dades que presenten diferents resultats. En concret, en l'àmbit d'aquesta pràctica estudiarem dues possibilitats:

Descomposició LU

```
Per a k = 0, 1, ..., n-2
  si akk = 0 aleshores abandona
  Per a i = k+1, ..., n-1
    aik = aik/akk
    Per a j = k+1, ..., n-1
      aij = aij - aik*akj
    Fper
  Fper
Fper
```

Eliminació Progressiva

```
Per a i = 0, 1, ..., n-1
  Per a j = i+1, ..., n-1
    bj = bj - lji*bi
  Fper
Fper
```

Substitució Regressiva

```
Per a i = n-1, ..., 0
  bi = bi/uii
  Per a j = i-1, ..., 0
    bj = bj - uji*bi
  Fper
Fper
```

Figura 3: Algorismes de descomposició LU i resolució de sistemes triangulars amb sobre-escriptura.

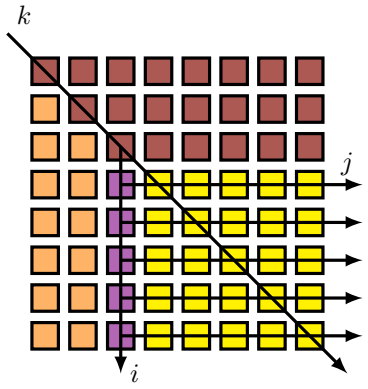

Figura 4: Esquema de la descomposició LU on s'il·lustra el recorregut dels tres bucles.

a) Distribució per blocs de files consecutives.

b) Distribució cíclica de les files.

Cada procés haurà de proporcionar la fila pivot en cas que la tinga o rebre-la del procés corresponent en un altre cas. Després d'açò, haurà d'actualitzar totes les files de les que disposa utilitzant la fila pivot.

Després de la factorització, s'haurà de realitzar la resolució dels sistemes triangulars. De forma similar, en cada iteració un procés disposarà de tota la informació per a calcular el valor de la component corresponent en el vector d'incògnites. Aquest valor haurà de ser propagat a tots els altres processos que encara tinguen files pendents perquè puguen actualitzar el vector de termes independents amb les contribucions en cada pas. Al final, tots els processos contenen el vector d'incògnites complet.

Atenent a cadascuna de les distribucions, s'hauran de tenir en compte les següents consideracions:

- *Blocs de files.* Distribució equilibrada del nombre de files entre els processos i vàlida per a qualsevol grandària de problema n. A cada procés, incloent P_0, se li assignarà un bloc de files consecutives.
- *Distribució cíclica per files.* Distribució equilibrada del nombre de files entre els processos i vàlida per a qualsevol grandària del problema n. A cada procés se li assignarà un bloc de files no consecutives i que es troben separades una distància igual al nombre de processos.

3 Tasca a realitzar

La versió per blocs de files es proporciona, per la qual cosa únicament s'ha d'implementar la versió cíclica.

Una vegada realitzada la versió cíclica, cal obtenir resultats experimentals de les dues variants, comparant les prestacions d'ambdues per a diferent nombre de processos i grandària de problema. Analitzar quina de les dues versions és més eficient i en quins casos.

Apèndix: codi de la versió per blocs

El codi de la versió paral·lela per blocs es mostra a continuació. Aquest programa implementa la generació de la matriu A i el vector b, el repartiment d'aquestes dades entre els diferents processos segons una distribució orientada a blocs de files, la factorització LU, la resolució del sistema d'equacions triangular inferior i la resolució del sistema d'equacions triangular superior, aquestes tres últimes en paral·lel. A més proporciona funcions per a reservar i alliberar memòria i per a mostrar les matrius i vectors per pantalla.

El codi consta, a més de la funció principal, de les següents funcions:

- `double **ppdGenMat(int nN, int nM)`. Funció que rep les dimensions d'una matriu i retorna un vector de punters que apunten a les diferents files d'una matriu. Aquesta funció realitza la reserva de memòria dinàmica.
- `int nRellenaMat(double **ppdMat, int nN, int nM)`. Funció que donada una matriu representada com un vector de punters i les seues dues dimensions, farcida aquesta matriu amb els valors d'una matriu de Toeplitz.
- `double *pdGenTI(int nN)`. Funció que donada una dimensió genera un vector de termes independents (en memòria dinàmica) de tal forma que la solució del sistema siga un vector tot a 1.
- `int nPrintMat(double **ppdMat, int nN, int nM)`. Funció que imprimeix de forma tabulada el contingut d'una matriu.
- `int nSustReg(double **ppdU, double *pdB, int nN, int nId, int nP)`. Funció que implementa el mètode de substitució regressiva per a resoldre un sistema d'equacions de `nN` files en el qual la matriu de coeficients és una matriu triangular superior. La solució es sobreescriu en el vector de termes independents. La funció s'executa sobre un sistema amb `nP` processos i cada procés ha de subministrar l'índex del procés `nId`.
- `int nLiberaMat(double **ppdMat)`. Funció que allibera la memòria reservada per a una matriu dinàmica.
- `int nPrintVec(double *ppdVec, int nN)`. Funció que mostra per pantalla el contingut d'un vector de dimensió `nN`.
- `int nElimProg(double **ppdL, double *pdB, int nN, int nId, int nP)`. Funció que implementa el mètode d'eliminació progressiva per a resoldre un sistema d'equacions de `nN` files en el qual la matriu de coeficients és una matriu triangular inferior unitat. La solució es sobreescriu en el vector de termes independents. La funció s'executa sobre un sistema amb `nP` processos i cada procés ha de subministrar l'índex del procés `nId`.
- `int nLU(double **ppdMat, int nN, int nId, int nP)`. Funció que implementa la factorització LU d'una matriu no singular, subministrada com a paràmetre i de dimensió `nN`. La L i la U sobreescriuen a la matriu d'entrada, de manera que la matriu L ocupa el triangle inferior (excepte la diagonal) i U el superior (incloent la diagonal). La funció s'executa sobre un sistema amb `nP` processos i cada procés ha de subministrar l'índex del procés `nId`.

```
/* Factoritzacio LU                                        */
/* Realitza la descomposicio LU sobre la matriu A          */
/* En el triangul inferior de A deixa la U i en el inf.    */
/* deixa la L. La digonal pertany a la U.                  */
/* Entrada: Matriu A i dimensio                            */
/* Eixida: Matriu L / U                                    */
int nLU(double **ppdMat, int nN, int nId, int nP, int nBloque) {

  int i,j,k;
  double *pdPivote;

  if (ppdMat == NULL)
    return -2;

  if ( (nN<=0) || (nN>MAX_DIM) )
    return -3;

  pdPivote = (double *)malloc(sizeof(double)*nN);

  for (k=0;k<nN-1;k++) {
    if (k/nBloque == nId) {
```

```
      if (fabs(ppdMat[k%nBloque][k])<EPSILON) {
        return -1;
      }
      memcpy(pdPivote, ppdMat[k%nBloque], nN*sizeof(double));
    }

    MPI_Bcast(pdPivote, nN, MPI_DOUBLE, k/nBloque, MPI_COMM_WORLD);

    for (i=k+1;i<nN;i++) {
      if (i/nBloque == nId) {
        ppdMat[i%nBloque][k] = ppdMat[i%nBloque][k] / pdPivote[k];
        for (j=k+1;j<nN;j++) {
          ppdMat[i%nBloque][j] -= ppdMat[i%nBloque][k] * pdPivote[j];
        }
      }
    }
  }
  free (pdPivote);
  return 0;
}

/* Eliminacio Progressiva                                */
/* Resol el sistema Lx = b                               */
/* Assumeix que la diagonal es la unitat                 */
/* Entrada: Matriu L, Vector B i Dimensio                */
/* Eixida: Vector Resultat                               */
/* Nota: Asumeix que la memoria esta reservada           */
/* Nota: Retorna la solucio en el vector B               */
int nElimProg(double **ppdL, double *pdB, int nN, int nId, int nP, int nBloque) {
  int i, j;

  if (ppdL == NULL)
    return -2;

  if ( (nN<=0) || (nN>MAX_DIM) )
    return -3;

  for(i=0;i<nN;i++) {
    MPI_Bcast(&pdB[i], 1, MPI_DOUBLE, i/nBloque, MPI_COMM_WORLD);
    for (j=i+1;j<nN;j++) {
      if (j/nBloque == nId) {
        pdB[j] -= ppdL[j%nBloque][i] * pdB[i];
      }
    }
  }
  return 0;
}

/* Sustitucio Regressiva                                 */
/* Resol el sistema Ux = b                               */
/* Entrada: Matriu U, Vector B i Dimensio                */
/* Eixida: Vector Resultat                               */
/* Nota: Asumeix que la memoria esta reservada           */
/* Nota: Retorna la solucio en el vector B               */
int nSustReg(double **ppdU, double *pdB, int nN, int nId, int nP, int nBloque) {
  int i, j;

  if (ppdU == NULL)
    return -2;

  if ( (nN<=0) || (nN>MAX_DIM) )
    return -3;

  for(i=nN-1;i>=0;i--) {
    if (i/nBloque == nId) {
```

```
      if (fabs(ppdU[i%nBloque][i])<EPSILON) return -1;
      pdB[i] = pdB[i] / ppdU[i%nBloque][i];
    }
    MPI_Bcast(&pdB[i], 1, MPI_DOUBLE, i/nBloque, MPI_COMM_WORLD);
    for (j=i-1;j>=0;j--) {
      if (j/nBloque == nId) {
        pdB[j] -= ppdU[j%nBloque][i] * pdB[i];
      }
    }
  }
  return 0;
}
```