Grundlagen der Parallelrechnerprogrammierung

(M. Pester)

Erweiterte Aufgabenstellung(en) für Abschlussbeleg

Es ist ein Programm zur parallelen Multiplikation einer $n \times n$-Matrix A mit einem Vektor x zu schreiben.
Mit dieser Operation soll ein Gleichungssystem $Ax = b$ mittels (einfachem) Iterationsverfahren gelöst werden. Auch das (Unter-)Programm zur parallelen Berechnung des Skalarproduktes ist dabei zu verwenden, um etwa einen Abbruchtest $\|Ax^{(k)} - b\| < \varepsilon$ durchzuführen.

Teilaufgaben / Hinweise zum ersten Teil (Matrix-Vektor-Multiplikation)

1. Die Matrix A soll auf `NPROC` Prozessoren verteilt gespeichert werden (`NPROC`$= 1, 2, 4, 8, \ldots$), indem jeder Prozessor seinen Teil von A selbst ausrechnet, zum Testen verwenden wir irgend eine Formel der Art

   $$a_{ij} = f(i, j, n).$$

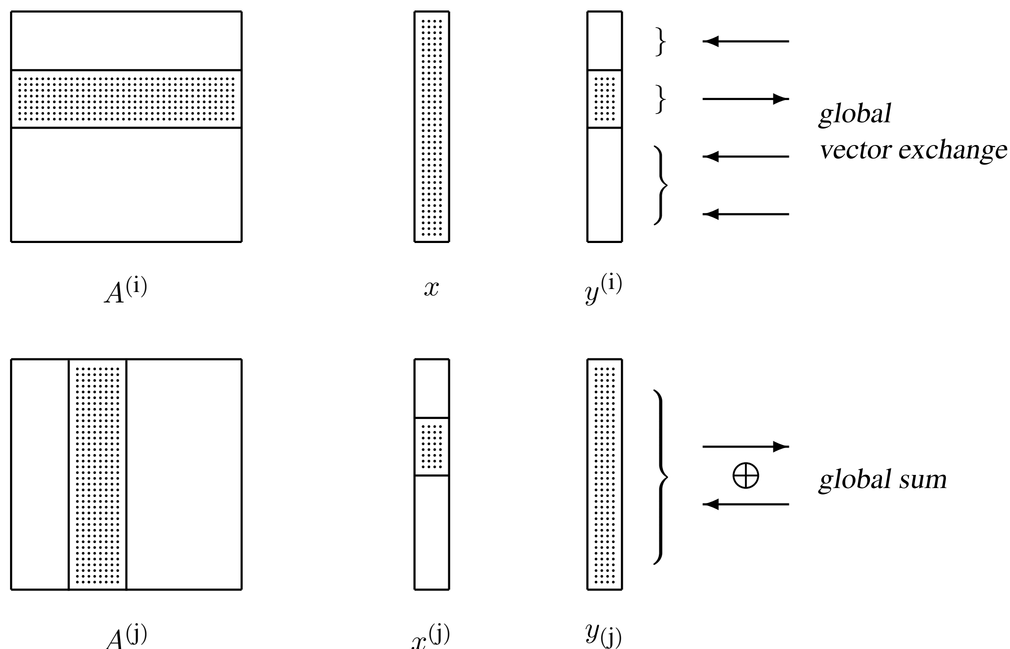
   Um ein „gutartiges" Verhalten des Iterationsverfahrens zu erreichen, sollte die Matrix (symmetrisch) positiv definit sein, also $a_{ij} = a_{ji}$ und z. B. Diagonaldominanz $a_{ii} > \sum_{j \neq i} |a_{ij}|$.

   Bekannt ist, dass die Hilbertmatrix

   $$\begin{array}{rcll} a_{ij} & = & 1.0/(i + j - 1.0), \quad i, j = 1, \ldots, n & \text{Indizes in Fortran, bzw.} \\ a_{ij} & = & 1.0/(i + j + 1.0), \quad i, j = 0, \ldots, n - 1 & \text{Indizes in C.} \end{array}$$

   ausgesprochen schlechte Eigenschaften besitzt. Man kann aber einen „großen" Wert zur Diagonale addieren (mindestens die Summe der restlichen Elemente der Zeile), um Diagonaldominanz zu erhalten. Die Belegung der Matrixelemente ist als Unterprogramm zu realisieren, das auf jedem Prozessor nur den dort zu speichernden Teil der Matrix berechnet. Dabei ist zu beachten, dass die globalen Indizes i und j von den im lokal gespeicherten Teil der Matrix für den Speicherzugriff benutzten Indizes abweichen.

2. Man entscheide sich für eine der folgenden Verteilungsvarianten für die Matrix A mit der zugehörigen Verteilung der Vektoren.



3. Die Verteilung der Elemente soll möglichst gleichmäßig auf alle Prozessoren erfolgen, auch wenn n kein ganzzahliges Vielfaches von `NPROC` ist. Aber im Interesse des geringeren Programmieraufwandes werden nur ganze Zeilen oder Spalten einem Prozessor zugeordnet, so dass jeder Prozessor eine $n \times m$- oder $m \times n$-Matrix verarbeitet. Speicherplatz für zwei Vektoren der Länge n ist ebenfalls auf jedem Prozessor vorzusehen.
   Die lokale Zeilen- bzw. Spaltenzahl m muss nicht auf jedem Prozessor gleich sein, aber es gilt $\sum m_{lokal} = n$.

   Falls n nicht durch die Anzahl `NPROC` der Prozessoren teilbar ist, d.h. $\texttt{NPROC} \cdot \left[\frac{n}{\texttt{NPROC}}\right] < n$, wählt man zunächst $m = \left[\frac{n}{\texttt{NPROC}}\right] + 1$ und hat nun die Wahl:

   - entweder Prozessor $\texttt{NPROC}-1$ ändert seine Anzahl auf $m := n - (\texttt{NPROC} - 1) \cdot m$ oder
   - die ersten k Prozessoren ($k = n - \texttt{NPROC} \cdot \left[\frac{n}{\texttt{NPROC}}\right]$), d.h. `ICH<k`, verwenden m, die anderen $m - 1$.

   Für die Benutzung der Kommunikationsroutine `CUBE_EXCH` ist die erste Variante günstiger, da hier jeder Prozessor ein Datenpaket gleicher Länge versenden muss (der letzte sendet dabei mehr Daten als er berechnet hat). Dann muss aber der Speicherplatz für die Vektoren jeweils „reichlich“ bemessen werden: $\texttt{NV}= p \cdot \left(\left[\frac{n}{p}\right] + 1\right)$.

   — vgl. genauere Hinweise ab Seite 4 —

Teilaufgaben / Hinweise zum zweiten Teil (iterativer Löser)

4. Um den Gleichungssystemlöser zu testen, gebe man sich eine Lösung $x = (x_1, \ldots, x_n)^\top$ vor und berechne die rechte Seite $b = Ax$ mit obiger Matrix-Vektor-Multiplikation.

   Als Iterationsverfahren kann z. B. das Jacobi-Verfahren (Gesamtschrittverfahren) benutzt werden:

   - Die Matrix A wird (in Gedanken!) zerlegt in $A = L + D + R$ (L - linke untere Dreiecksmatrix, D - Diagonale, R - rechte obere Dreiecksmatrix).
   - Aus $Ax = b$ bzw. $(L + D + R)x = b$ wird mit $Dx = b - (L + R)x$ eine Iterationsvorschrift
     $$x^{(k+1)} = D^{-1} \cdot (b - (L + R)x^{(k)}).$$
   - Da $(L + R)$ fast mit A übereinstimmt (nur mit $a_{ii} = 0$), soll dafür kein zusätzlicher Speicherplatz verschwendet werden.
     Entweder die Diagonalelemente von A werden auf einen Vektor d umgespeichert und in A hart auf 0 gesetzt
     oder für die Multiplikation $(L + R)x$ wird eine modifizierte Variante der obigen Matrix-Vektor-Multiplikation verwendet, bei der die Diagonalelemente „ignoriert“ werden.
   - **Achtung:** Da jeder Prozessor nur einen Teil der Matrix A besitzt, findet er auch seinen Teil der Diagonalelemente jeweils an einer anderen Stelle der lokalen Rechteckmatrix!
   - Der Algorithmus könnte dann folgendermaßen ablaufen:
     - $y := b - (L + R)\check{x}$ (als „erweiterte“ Matrix-Vektor-Multiplikation oder in zwei Schritten)
     - $\hat{x} := D^{-1}y$ (lediglich: $x_i := y_i/a_{ii}$)
     - $s := \hat{x} - \check{x}$ (und teste $\langle s, s\rangle < \varepsilon^2$)
     - nächster Iterationsschritt mit $\check{x} := \hat{x}$

   Prüfen Sie, welche dieser Operationen rein lokal auf jedem Prozessor ausgeführt werden können und wann ein Datenaustausch erforderlich wird.

5. Die Matrix A ist ausschließlich lokal zu speichern. Bei den Vektoren reichen oft nur die zum lokalen Teil der Matrix gehörenden Komponenten. Es ist zu prüfen, für welche Vektoren zu welchem Zeitpunkt im Algorithmus ein globaler Austausch notwendig ist.

   Um die Daten richtig zu verteilen und zu verarbeiten, ist zu beachten:

- in Fortran werden Matrizen spaltenweise, in C aber zeilenweise gespeichert, falls Arrays mit zwei Indizes dafür benutzt werden.

Das könnte interessant werden, wenn man die Matrixmultiplikation für die lokal gespeicherten Teilmatrizen durch Aufrufe von Unterprogrammen wie Skalarprodukt `rscapr(...) / dscapr(...)` oder durch Linearkombination `vsaxpy(...) / vdaxpy(...)` von Vektoren (Spalten/Zeilen) realisiert. (vgl. Dokumentation[1], S. 4/5)

6. Der Nutzerdialog auf Prozessor 0 umfasst
   - Eingabe der Dimension n und der Abbruchgenauigkeit ε
   - Ausgabe der Abweichung der berechneten Näherungslösung von der im Programm vorgegebenen und Ausgabe der Anzahl der dazu benötigten Iterationen.

7. **Zeitmessung** ist zwar auf einem Multi-User-System nicht viel wert, aber man kann ja trotzdem mal messen, wie lange es dauert und welche Relationen zwischen CPU-Zeit und Kommunikationszeit bestehen ⇒ Dokumentation[1], S. 25.

8. **Hinweise zur Implementierung** (mit den „Chemnitzer Bibliotheken“):

   Hauptprogramm:

```
      (Fortran)                         (C)
...                                ...
include 'include/trnet.inc'        #include "trnet.h"
                                   #include "vbasmod.h"
...                                ...
DOUBLE PRECISION eps               double eps
...                                ...
call trinit(0)                     int Lng, Null=0;
                                   trinitc (&Null);
if (ICH .EQ. 0) then               if ( ICH==0 ) {
         ...      Eingabe-Dialog       ...
endif                              }
call Tree_Down_0(2,eps)            Lng=2; tree_down_0(&Lng,&eps);
call Tree_Down_0(1,n)              Lng=1; tree_down_0(&Lng,&n);
...                                ...
      (Aufruf der Unterprogramme zur Matrixberechnung und
       Festlegung des Startvektors x0)
...                                ...
      (Iteration als Unterprogramm aufrufen)
...                                ...
if (ICH .EQ. 0) then               if ( ICH==0 ) {
          ...      Ausgabe-Dialog       ...
endif                              }
call trclose                       trclose();
```

Nützliche Compileroptionen (speziell für die Linux-GNU-Compiler)

```
   gfortran -ffast-math  ...
bzw.
   gcc -D UNDER -ffast-math  ... \
       -I ${SFB}/FEM/libs/headers ...
```

Weitere für diese Aufgabe hilfreiche Bibliotheksroutinen:

```
cube_exch(...)                                    (S.16)
cube_doi(...)   oder   cube_dod(...)              (S.19)
ring_out(...) / ring_receive0(...)                (S.22)
...
```

[1] http://www.tu-chemnitz.de/sfb393/Files/PDF/sfb02-01.pdf

9. Beim Linken des Programms sind dann die entsprechenden Bibliotheken mit anzugeben:

   Für die Nutzung von PVM:

   ```
   gfortran ... -L/afs/tu-chemnitz.de/project/sfb393/FEM/libs/LINUXX \
             -lGraf -lCubecom -lTools -lvbasmod -lfpvm3 -lpvm3 \
             -lX11
   ```

   bzw. für MPI:

   ```
   mpif77 ..... -L/afs/tu-chemnitz.de/project/sfb393/FEM/libs/LINUXX \
             -lGraf -lMPIcubecom -lTools -lvbasmod \
             -lX11
   ```

   Die `X11`-Bibliothek und `-lGraf` werden nur im Falle der Zeitmessung und -Auswertung benötigt.

   (Tipp: Ein **Makefile** erspart das wiederholte Eintippen langer Kommandos zum Compilieren.)

Zur Implementierung der parallelen Matrix-Vektor-Multiplikation

Aus globaler Sicht: $y := Ax, \quad A \in \mathbb{R}^{n,n},\ x, y \in \mathbb{R}^n, \quad y_i = \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, \ldots, n.$

Die lokale Sicht des Prozessors `ICH` (von insgesamt `NPROC` Prozessoren):

Variante 1:

$$\tilde{y} := \tilde{A}x, \quad \tilde{A} \in \mathbb{R}^{m,n},\ x \in \mathbb{R}^n,\ \tilde{y} \in \mathbb{R}^m, \quad \tilde{y}_i = \sum_{j=1}^{n} \tilde{a}_{ij} x_j, \quad i = 1, \ldots, m$$

$$y := \begin{bmatrix} \tilde{y}^{(0)} \\ \vdots \\ \tilde{y}^{(\texttt{NPROC-1})} \end{bmatrix}, \qquad \text{globaler Austausch: } \texttt{cube_exch(2*m0,y,y,ICH)}\ ,$$

Die Längenangabe erfolgt hier in Worten, deshalb `2*m0`, wenn m_0 `double`-Elemente auszutauschen sind. Vor der Operation enthält `y` den lokalen Vektor der Länge m_0, danach den globalen Vektor der Länge n.

Die Länge des lokalen Vektors $\tilde{y}$ muss auf jedem Prozessor gleich sein, um die (einfachere) Kommunikationsroutine `cube_exch` anwenden zu können. So kann man die Daten z.B. folgendermaßen aufteilen:

$$\begin{aligned} m_0 &= \left\lceil \frac{n + \texttt{NPROC} - 1}{\texttt{NPROC}} \right\rceil, \\ m &= \begin{cases} m_0, & \text{falls } \texttt{ICH} < \texttt{NPROC} - 1, \\ n - m_0 \cdot (\texttt{NPROC} - 1), & \text{falls } \texttt{ICH} = \texttt{NPROC} - 1. \end{cases} \end{aligned}$$

Zum Rechnen wird dann m, und zum Kommunizieren m_0 benutzt, so dass der Vektor y Platz für $(m_0 \cdot \texttt{NPROC}) \geq n$ Elemente haben muss, von denen aber nur n weiterverwendet werden.

Den unteren und oberen Zeilenindex bezüglich der gesamten Matrix berechnet man daraus lokal:

$$\begin{aligned} \texttt{iu} &= m_0 \cdot \texttt{ICH} + 1, \\ \texttt{io} &= \texttt{iu} + m - 1. \end{aligned}$$

Die globalen Indizes benötigt man z.B., um die Matrixelemente lokal auf dem jeweiligen Prozessor zu berechnen:
Wenn global gilt $a_{ij} = f(i, j, n)$, so gilt für den lokal gespeicherten Teil $\tilde{A}$ der Matrix A:
$\tilde{a}_{ij} = f(\texttt{iu} + i - 1, j, n)$.

Variante 2: $[\,]\cdot[\,]=[\,]$

$$\tilde{y} := \tilde{A}\tilde{x}, \quad \tilde{A} \in \mathbb{R}^{n,m},\ x \in \mathbb{R}^m,\ \tilde{y} \in \mathbb{R}^n, \quad \tilde{y}_i = \sum_{j=1}^{m} \tilde{a}_{ij}\tilde{x}_j, \quad i = 1,\ldots,n$$

$$y := \tilde{y}^{(0)} + \ldots + \tilde{y}^{(\mathtt{NPROC-1})}, \qquad \text{globale Summe: } \mathtt{cube_dod(n,y,y,H,vdplus)},$$

(globale Summe benötigt einen Hilfsvektor `H` der Länge n).

Hier kann die lokale Spaltenanzahl m wie bei Variante 1 oder auch folgendermaßen bestimmt werden:

$$\begin{aligned} m_0 &= \left[\frac{n}{\mathtt{NPROC}}\right], \\ m_1 &= n - m_0 \cdot \mathtt{NPROC} \quad (= 0, \text{ falls } n \text{ durch } \mathtt{NPROC} \text{ teilbar}), \\ m &= \begin{cases} m_0, & \text{falls } \mathtt{ICH} \geq m_1, \\ m_0 + 1, & \text{falls } \mathtt{ICH} < m_1. \end{cases} \end{aligned}$$

(Die ersten m_1 Prozessoren erhalten also je eine Zeile mehr.)

Damit lässt sich auch der untere und obere (globale) Spaltenindex des lokalen Matrixblocks bestimmen:

$$\begin{aligned} \mathtt{iu} &= \begin{cases} m \cdot \mathtt{ICH} + 1, & \text{falls } \mathtt{ICH} < m_1, \qquad (m = m_0 + 1) \\ m \cdot \mathtt{ICH} + m_1 + 1, & \text{falls } \mathtt{ICH} \geq m_1, \qquad (m = m_0) \end{cases} \\ \mathtt{io} &= \mathtt{iu} + m - 1 \end{aligned}$$

In C werden Indizes von $0,\ldots,n-1$ (statt $1,\ldots,n$) verwendet. Deshalb sind einige der obigen Ausdrücke entsprechend anzupassen.