Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Prof. Dr. Christian Engwer, MSc. Marcel Koch

14.12.2017

Übung zur Vorlesung

Wissenschaftliches Rechnen

WS 2017/18 — Blatt 10

Abgabe: 21.12.2017, 10:00 Uhr, Briefkasten 111
Code zusätzlich per e-mail an `marcel.koch@uni-muenster.de`

Aufgabe 1 (Leistungsanalyse, skalierter Speedup) (4 Punkte)

Betrachten Sie das parallele Programm zur Berechnung des Skalarproduktes, welches Sie in der Vorlesung kennengelernt haben.

(a) Erläutern Sie die Formeln für

- die Laufzeit $T_S(N)$ des besten sequentiellen Algorithmus' (auf einem Kern des Parallelrechners),
- die Laufzeit $T_P(N, P)$ des parallelen Algorithmus',
- den Speedup $S(N, P)$.

(b) Untersuchen Sie, wie sich der Speedup in Abhängigkeit von P verhält. Skalieren Sie ihn dazu durch die Berechnung geeigneter Problemgrößen

- $N = N_A$,
- $N = N_G(P)$,
- $N = N_M(P)$,
- $N = N_I(P)$

so, dass sich eine feste sequentielle Ausführungszeit (d.h. $T_S(N_A) = T_{\text{fix}}$), eine feste parallele Ausführungszeit (d.h. $T_P(N_G(P), P) = T_{\text{fix}}$), ein fester Speicherbedarf pro Prozessor (d.h. $M(N_M(P)) = M_0 P$) oder eine konstante Effizienz (d.h. $E(N_I(P), P) = E_0$) ergibt. Ermitteln Sie jeweils den resultierenden Speedup

- $S_A(P) = S(N_A, P)$,
- $S_G(P) = S(N_G(P), P)$,
- $S_M(P) = S(N_M(P), P)$,
- $S_I(P) = S(N_I(P), P)$.

(c) Bestimmen Sie jeweils die Komplexitätsklasse $\Theta(f)$, $f : \mathbb{N} \to \mathbb{R}$ mit $P \mapsto f(P)$, des in Aufgabenteil (b) ermittelten Speedups und vergleichen Sie die Größenordnungen für die verschiedenen Skalierungen.

Hinweis: Bei der Bestimmung von $S_G(P)$ dürfen Sie davon ausgehen, dass die feste parallele Ausführungszeit wesentlich größer ist als die Zeit t_a für eine arithmetische Operation und die Zeit t_m zum Versenden einer Fließkommazahl.

Aufgabe 2 (Implementierung der FEM zur Lösung des Poisson-Problems) (12 Punkte)

Basierend auf der Programmmierübung von Blatt 9 implementieren Sie nun einen Finite Elemente Löser für das Poisson-Problem mit Dirichlet Randwerten,

$$\begin{aligned} -\Delta u &= f \text{ in } \Omega, \\ u &= g \text{ auf } \partial\Omega. \end{aligned}$$

In der schwachen Formulierung ist nun ein $u \in H^1$ mit $u|_{\partial\Omega} = g$ gesucht so, dass

$$\begin{aligned} \int_\Omega \nabla u \cdot \nabla v \, \mathrm{dx} &= \int_\Omega f v \, \mathrm{dx} \quad \forall v \in H_0^1. \\ \Rightarrow a(u,v) &= F(v) \end{aligned}$$

Wir betrachten zunächst nur den Fall $g = 0$, d.h. $u \in H_0^1$. Der allgemeine Fall wird zum Schluss behandelt. Zur Diskretisierung des Problems wird das Gebiet Ω durch $\mathcal{T}(\Omega)$ in Dreiecke partitioniert. Die Lösung u wird nun in dem endlich dimensionalen Unterraum

$$V_h = \{v_h \in C(\bar{\Omega}) \mid v_h|_E \in P_1(E), \forall E \in (T)(\Omega)\}$$

approximiert. Die Approximationu_h lässt sich schreiben als

$$u_h = \sum_{i=0}^{N} \varphi_i x_i,$$

wobei $N = \dim V_h$, $\{\varphi_i\}_{i=0}^N$ eine Basis von V_h ist und $x = (x_i)_{i=0}^N \in \mathbb{R}$. Als Basis von V_h wählen wir die Knotenbasis, d.h. für einen Knoten der Partitionierung a_j gilt $\varphi_i(a_j) = \delta_{ij}$. Damit lässt sich das lineare System

$$Ax = b$$

aufstellen mit

$$\begin{aligned} A_{ij} = a(\varphi_j, \varphi_i) &= \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, \mathrm{dx} = \sum_{E\in\mathcal{T}} \int_E \nabla\varphi_j \cdot \nabla\varphi_i \, \mathrm{dx}, \\ b_i = F(\varphi_i) &= \int_\Omega f\varphi_i \, \mathrm{dx} = \sum_{E\in\mathcal{T}} \int_E f\varphi_i \, \mathrm{dx}. \end{aligned}$$

Die Basisfunktionen φ_i haben nur einen sehr beschränkten Träger, daher ist es sinnvoll, das Integral über das gesamte Gebiet nur auf die Dreiecke $E \in \mathcal{T}$ mit $\mathrm{supp}(\varphi_i) \cap E \neq \emptyset$ zu beschränken. Weiter lässt sich zu jedem Dreieck E eine Abbildung $T_E : \hat{E} \mapsto E$ definieren, die das Einheitsdreieck $\hat{E} = \{(x,y) \in \mathbb{R}^2 | x, y \geq 0,\ x+y \leq 1\}$, auf das Dreieck E der Partition abbildet. Auf dem Einheitsdreieck definieren wir die drei linearen Knotenbasisfunktionen

$$\begin{aligned} \varphi_0(x,y) &= 1 - x - y, \\ \varphi_1(x,y) &= x, \\ \varphi_2(x,y) &= y. \end{aligned}$$

Zudem existiert eine Abbildung P_E, die dem Index j eines Knoten im Einheitsdreieck, den Index $P_E(j)$ zuweist so, dass $T_E(a_j) = a_{P_E(j)}$. Eine Basisfunktion φ_i hat damit die Darstellung

$$\varphi_i|_E(x) = \hat{\varphi}_{P_E(i)^{-1}}(T_E^{-1}(x))$$

auf einem Element E, das den Träger von φ_i enthält. Der Index der Basisfunktion $\hat{\varphi}_j$ ist so sogewählt, dass für den Knoten $\hat{a}_j$ des Einheitsdreiecks gilt $T_E(\hat{a}_j) = a_i$. Damit lässt sich jedes Integral über ein Element E auf ein Integral über $\hat{E}$ schreiben:

$$\int_E \nabla\varphi_j \cdot \nabla\varphi_i \,\mathrm{dx} = \int_{\hat{E}} (\mathrm{D}T_E^{-T}\, \nabla\hat{\varphi}_{P_E(j)^{-1}}) \cdot (\mathrm{D}T_E^{-T}\, \nabla\hat{\varphi}_{P_E(i)^{-1}}) |\det \mathrm{D}T_E| \,\mathrm{dx}$$
$$\int_E f\varphi_i \,\mathrm{dx} = \int_{\hat{E}} f(T_E(x))\hat{\varphi}_{P_E(i)^{-1}} |\det \mathrm{D}T_E| \,\mathrm{dx}$$

Die Einträge der Matrix A und der rechten Seite b lassen sich nun berechnen als

$$A_{ij} = \sum_{\substack{E\in\mathcal{T} \\ \mathrm{supp}(\varphi_i)\cap\mathrm{supp}(\varphi_j)\cap E\neq\emptyset}} \int_{\hat{E}} (\mathrm{D}T_E^{-T}\, \nabla\hat{\varphi}_{P_E(j)^{-1}}) \cdot (\mathrm{D}T_E^{-T}\, \nabla\hat{\varphi}_{P_E(i)^{-1}}) |\det \mathrm{D}T_E| \,\mathrm{dx}$$
$$b_i = \sum_{\substack{E\in\mathcal{T} \\ \mathrm{supp}(\varphi_i)\cap E\neq\emptyset}} \int_{\hat{E}} f(T_E(x))\hat{\varphi}_{P_E(i)^{-1}} |\det \mathrm{D}T_E| \,\mathrm{dx}$$

Anstatt diese Einträge direkt zu berechnen, wird über alle Elemente des Gitters iteriert und die Kontributionen pro Element an die richtigen Stellen addiert, d.h.

$$A_{P_E(i),P_E(j)} = A_{P_E(i),P_E(j)} + \int_{\hat{E}} (\mathrm{D}T_E^{-T}\, \nabla\hat{\varphi}_j) \cdot (\mathrm{D}T_E^{-T}\, \nabla\hat{\varphi}_i) |\det \mathrm{D}T_E| \,\mathrm{dx}, \tag{1a}$$
$$b_{P_E(i)} = b_{P_E(i)} + \int_{\hat{E}} f(T_E(x))\hat{\varphi}_i |\det \mathrm{D}T_E| \,\mathrm{dx}, \tag{1b}$$

für $i, j = 0, 1, 2$ und alle $E \in \mathcal{T}$. Die lokalen Kontributionen pro Element, lassen sich auch als 3×3 Matrizen oder 3 dimensionale Vektoren zusammenfassen. Man spricht dann von der *lokalen Steiffigkeitsmatrix* und der *lokalen rechten Seite*.

- Als erstes muss dazu die Gitterdatenstruktur angepasst werden, um die Randknoten identifizieren zu können. Speichern Sie sich für jedem Knoten zu welchem Randtypen er gehört. Unterscheiden Sie nur zwischen `Inner`, für innere Knoten, und `Dirichlet`, für Dirichlet Randknoten. Das Gitterformat ist nun wie folgt gegeben:

```
NNODES number_of_nodes //=N
x_0 y_0 bc_type_0 //bc_type==0 ==> Innerer Knoten
x_1 y_1 bc_type_1 //bc_type==1 ==> Dirichlet Knoten
.
.
.
x_{N-1} y_{N-1} bc_type_{N-1}
NELEMENTS number_of_elements //=E
i_0 j_0 k_0
i_1 j_1 k_1
```

```
11      .
        .
13      .
        i_{E-1}  j_{E-1}  k_{E-1}
```

Sie müssen also auch die Einlesefunktion des Gitters anpassen. Den Randtypen können Sie zum Beispiel als `enum`

```
enum BC_TYPE {INNER, DIRICHLET};
```

oder als eigenständige Klassen implementieren.

- Um die lokalen Beiträge zu berechnen müssen Sie die Basisfunktionen $\hat{\varphi}_i$ und deren Gradienten auswerten. Schreiben Sie dazu eine Klasse `P1LocalBasis` die folgende Methoden implementiert:
  - `VectorType eval(const Coord&)` wertet alle Basisfunktionen an der Stelle x in $\hat{E}$ aus, wobei `VectorType` einen 3 dimensionalen Vektor darstellt und von Ihnen passend zu wählen ist,
  - `MatrixType evalGradient(const Coord&)` wertet die Gradienten aller Basisfunktionen an der Stelle x in $\hat{E}$ aus, wobei `MatrixType` eine 2×3 Matrix darstellt und von Ihnen passend zu wählen ist,
  - `std::size_t size()` liefert die Anzahl an Basisfunktionen.
- Die Berechnungen in (1) auf dem Einheitsdreieck sollen in einer Klasse `P1LocalOperator` durchgeführt werden. Schreiben Sie dazu eine Klasse `P1LocalOperator` die folgende Methoden und Attribute implementiert:
  - ein Funktionsobjekt, welches die Funktion f darstellt,
  - `MatrixType localStiffnessMatrix(const P1Triangle&)` liefert die lokale Steiffigkeitsmatrix nach (1) zurück, wobei `MatrixType` eine 3×3 Matrix darstellt und von Ihnen passend zu wählen ist (unabhängig von oben),
  - `VectorType localRHS(const P1Triangle&)` liefert die lokale rechte Seite nach (1) zurück, wobei `VectorType` einen 3 dimensionalen Vektor darstellt und von Ihnen passend zu wählen ist (unabhängig von oben).

  Zur Berechnung der Integrale benutzen Sie die Gauß-Quadraturformel mit einem Quadraturpunkt, d.h.
  $$\int_{\hat{E}} h(x)\ \mathrm{dx} \approx \frac{1}{2} h(1/3, 1/3).$$
- Schreiben Sie eine Funktion

  ```
  std::pair<Eigen::SparseMatrix<double>,Eigen::VectorXd>
  assembleSystem(const P1Grid&, const P1LocalOperator&);
  ```

  die die globale Steiffigkeitsmatrix und die globale rechte Seite aufstellt. Dazu iterieren Sie über das Gitter, berechnen zu jedem Element die lokalen Größen und addieren diese auf

die entschprechenden Einträge in der globalen Steiffigkeitsmatrix und rechte Seite. Dies geschieht analog zu der Programmieraufgabe zu Sparse-Matrizen auf dem Übungsblatt 9. Hierbei beachten Sie noch nicht mögliche Randwerte.

- Schreiben Sie eine Funktion

```
void incorporateDirichletBoundary(const P1Grid&, Eigen::SparseMatrix<double>&,
                                  Eigen::VectorXd&);
```

die die Dirichlet Randbedingung in das System einbaut. Dazu müssen Sie die Steiffigkeitsmatrix, sowie die rechte Seite verändern. Zunächst teilen wir die Lösung u_h auf in $u_h = u^0 + u^D$, wobei $u^0 \in H_0 1$ und $u^D \in H^1$ mit $u^D|_{\partial\Omega} = g$. Damit folgt für u^0

$$a(u^0, v) = F(v) - a(u^D, v),$$

also erhalten wir das System

$$a(\varphi_j, \varphi_i)x_i^0 = F(\varphi_i) - a(u^D, \varphi_i) \quad \forall i, j : a_i, a_j \in \Omega,$$

mit dem Koeffizientenvektro x^0 von u^0. Damit muss also nur noch für die Freiheitsgrade im inneren ein lineares Gleichungssystem gelöst werden, mit einer modifizierten rechten Seite. Die neue rechte Seite lässt sich durch

$$\tilde{b}_i = \begin{cases} b_i - a(u^D, \varphi_i) = b_i - \sum_{j=0}^{N} a(\varphi_j, \varphi_i)x_j^D & \forall i : a_i \in \Omega \\ g(a_i) \quad \forall i : a_i \in \partial\Omega \end{cases}$$

berechnen, wobei x^D der Koeffizientenvektor von u^D ist, also $x_i^D = g(a_i)$ falls $a_i \in \partial\Omega$ und $x_i^D = 0$ sonst. Mit der vorher erstellten Steiffigkeitsmatrix gilt dann

$$\tilde{b} = b - Ax^D.$$

Zusätzlich modifiziert man die Systemmatrix

$$\tilde{A}_{ij} = \begin{cases} 1 & i = j \text{ und } a_i \in \partial\Omega, \\ 0 & i \neq j \text{ und } a_i \in \partial\Omega, \\ 0 & i \neq j \text{ und } a_j \in \partial\Omega, \\ A_{ij} & \text{sonst}, \end{cases}$$

Damit erhält man das System

$$\tilde{A}x = \tilde{b}$$

- Auf den Gittern `p1structured_coarse` und `p1structured_fine` lösen Sie das Poisson-Problem mit

$$f(x, y) = -20(y(y-1) + x(x-1)),$$
$$g(x, y) = 0,$$

welches die exakte Lösung

$$u(x, y) = 10x(1-x)y(1-y)$$

hat. Plotten Sie die Lösung und den Fehler mit dem zur Verfügung gestellten VTKWriter.

- Auf dem Gitter `l-shape` lösen Sie das Poisson-Problem mit

$$f(x, y) = 0$$

und

$$g(x, y) = \begin{cases} 0 & x \in (0, 1) \text{ oder } y \in (0, 1) \\ 5 & y = 1, x \in (0, 0.25) \text{ oder } x = 1, y \in (0, 0.25) \\ 10 & (1 - x)^2 + (1 - y)^2 = 0.75^2, x, y \in (0.25, 1) \end{cases}$$

und plotten Sie das Ergebnis.