Übungen zur Vorlesung

Paralleles Höchstleistungsrechnen

Dr. S. Lang

Abgabe: 19. Dezember 2013 in der Übung

Übung 17 MPI: Kommunikation im Ring (5 Punkte)

Mit dieser Aufgabe wollen wir die ersten Schritte mit MPI wagen. Implementieren Sie eine Kommunikation von 8 Prozessen im Ring. Jeder Prozess soll als Nachricht seinen Rang einmal im Ring verschicken und terminieren, falls er wieder seinen Rang als Nachricht gesendet bekommt. Verwenden Sie synchronisierendes Senden bzw. Empfangen und eine der in der Vorlesung vorgestellten Techniken, um Verklemmungen zu vermeiden, z.B. Färben der Kanten. Jeder Prozess soll in jedem Empfang/Sende-Schritt seinen Rang und die soeben empfangene Nachricht ausgeben. Testen Sie Ihr Programm im Pool und geben Sie eine Ausgabe des Kommunikations-Ablaufs ab.

Genauere Informationen zur Verwendung von MPI im Pool finden auf der Homepage. Hilfreich zur Syntax sind die Manpages zu MPI (z.B. `man MPI_Comm_rank`).

Übung 18 Paralleles Berechnen von π mit MPI (5 Punkte)

Aus der Identität $\pi = 4(\arctan 1)$ erhält man durch Ausnutzen der Ableitung des arctan, $(\arctan x)' = 1/(1+x^2)$, eine Vorschrift zur Berechnung von π:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx.$$

Durch Teilen des Intervalls in n äquidistante Teilstücke kann das Integral mit der Mittelpunktsregel ausgewertet werden. Ein sequentielle Programm finden Sie in der Datei `piseq.c` auf der Homepage. Wir wollen es mit MPI parallelisieren. Die Strategie ist:

- Prozess 0 liest die Anzahl der Teilintervalle ein und teilt sie allen anderen Prozessen mit,
- Die `for`-Schleife über die Teilintervalle wird parallelisiert, jeder Prozess berechnet eine lokale Teilsumme. Die Ergebnisse werden von Prozess 0 mit einer Reduktions-Operation `MPI_Reduce` eingesammelt und die Teilsummen addiert.

Bestimmen Sie zunächst mit dem sequentiellen Prgramm die Konvergenzordnung der Mittelpunktsregel. Erstellen Sie dazu einen doppel-logarithmischen Plot, in dem Sie den Fehler in der Integration über der Intervallllänge h auftragen, die Steigung der Geraden ergibt die Ordnung. Implementieren Sie dann die parallele Version. Vergleichen Sie die Genauigkeit in den Rechnungen (Nachkommastellen) mit der sequentiellen Lösung und dem genauen Wert (kurze Diskussion).

Freiwillige Zusatzaufgabe

Die Anzahl der der gültigen Nachkommastellen kann durch Verwenden der *Gnu Multiprecision Arithmetic Library* (GMP, zu finden unter www.gmplib.org, für die meisten Linux-Distributionen gibt es ein Paket) erhöht werden. Somit könnte π z.B. bis zur 40., 60. oder 80. Stelle berechnet werden. Allerdings ist das gewählte Verfahren mit quadratischer Konvergenz dazu viel zu langsam, d.h. eine in der 80. Nachkommastelle exaktes Programm würde ewig rechnen. Implementieren Sie eine Version des sequentiellen oder parallelen Programms, die die GMP verwendet. Wenn Sie oben angesprochene Genauigkeiten erreichen wollen, müssen Sie ein besseres Verfahren als die Mittelpunktsregel verwenden. Anderenfalls testen Sie, bis zu welcher Genauigkeit Sie mit der GMP und Mittelpunktsregel in sinnvoller Rechenzeit kommen können. Den Referenz-Wert von *pi* können Sie im Netz besorgen.

Übung 19 Einfache Parallelisierung des Jacobi-Verfahrens mit MPI (10 Punkte)

Für lineare Gleichungssysteme $A\mathbf{x} = \mathbf{b}$, $A \in \mathbb{R}^{n\times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$, $n \in \mathbb{N}$, sind direkte Lösungsverfahren für große n meist ineffizient. Daher verwendet man oft iterative Verfahren wie das *Jacobi-Verfahren.* Man erhält es durch Aufspalten der Systemmatrix A in die obere und untere Dreicksmatrix U und L sowie die Diagonalmatrix D, $A = D + L + U$. Dies führt auf die Fixpunktgleichung $\mathbf{x} = D^{-1}(\mathbf{b} - (A - D)\mathbf{x})$, die unter gewissen Vorraussetzungen mit einer Fixpunktiteration gelöst werden kann: $\mathbf{x}^{(m+1)} = D^{-1}(\mathbf{b} - (A - D)\mathbf{x}^{(m)})$, Index $m \in \mathbb{N}$ indiziert den Iterationsschritt. Die i.te Gleichung für den $(m + 1)$. Schritt des Verfahrens lautet dann

$$x_i^{(m+1)} = \frac{b_i - \sum_{j\neq i} a_{ij} x_j^{(m)}}{a_{ii}}.$$

Als Startvektor $\mathbf{x}^0$ kann man jeden beliebigen Vektor verwenden. Die Abbruchbedingung ergibt sich aus der Berechung des Residuums $\mathbf{r}$ im m.ten Schritt, $\mathbf{r}^m := \mathbf{b} - A\mathbf{x}^{(m)}$: Ist eine geeignete Norm des Residuums (z.B. die Maximumsnorm $||\mathbf{r}||_\infty$) kleiner einer gegebenen Toleranz $\epsilon \in \mathbb{R}_+$, wird abgebrochen. Die Berechnungen einer Iteration hängen nur von der vorhergehenden Lösung ab und es gibt es keine Datenabhängigkeiten zwischen den neu zu berechnenden Werten x_i. Daher ist das Jacobi-Verfahren leicht zu parallelisieren.

Wir wollen das Jacobi-Verfahren verwenden, um die Diskretisierung der Poisson-Gleichung

$$\begin{aligned} -\triangle u &= f \quad \text{in } \Omega = (0, r)^2, \\ u &= 0 \quad \text{auf } \partial\Omega \end{aligned}$$

auf einem Quadrat mit Seitenlänge r zu berechnen. Die Quelle f sei $f(x, y) := 2\pi^2 \sin(\pi x) \cdot \sin(\pi y)$. Damit ist die analytische Lösung gegeben durch $u(x, y) = \sin(\pi x) \cdot \sin(\pi y)$. Das Einheitsquadrat wird von einem Gitter aus n^2 Punkten überzogen, siehe Abbildung 0.5 links mit $r = 4$ und $n = 9$. Der Abstand h zwischen zwei Punkten (die „Gitterweite“) ist $h = r/(n - 1)$. In der Abbildung rechts ist die Quelle f aufgetragen.

Ein Gleichungssystem in Matrixform erhält man durch Approximation der zweiten Ableitungen durch einen zentralen Differenzen-Quotienten an jedem inneren Gitterpunkt (i, j) mit $i, j = 1 \ldots n-1$. An den Randpunkten mit $i = 0$ oder $i = n$ (oder für j) ist die Lösung fest vorgegeben. Die Gitterpunkte kann man auch konsekutiv durch $k = i \cdot n + j$ nummerieren, es wird also jedes Indexpaar (i, j) auf einen einzigen Index k eineindeutig abgebildet. Dann kann man die Gitterfunktion

$$u^h := (u_{01}, u_{02}, \ldots, u_{0n}, u_{11}, \ldots, u_{1n}, \ldots, u_{n1}, \ldots u_{nn})^T = (u_1, \ldots u_N)^T$$

mit $N = n^2$ betrachten. Nach der Approximation der zweiten Ableitung lautet das Gleichungssystem für die unbekannte Gitterfunktion u^h (Näheres zur Herleitung in Kürze):

$$\frac{1}{h^2}\begin{pmatrix} 4 & -1 & \cdots & -1 & & & & & \\ -1 & 4 & -1 & \cdots & -1 & & & & \\ & \ddots & \ddots & \ddots & & \ddots & & & \\ & & & \ddots & -1 & & \ddots & & \\ & & & -1 & 4 & -1 & \cdots & -1 & \\ & & & & -1 & \ddots & & & \\ & & & & & & \ddots & -1 & \\ & & & & \cdots & & -1 & 4 & \end{pmatrix}_{N\times N} \cdot \begin{pmatrix} u_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} f_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_N \end{pmatrix}.$$

Die Werte f_1 bis f_N sind die Auswertungen der Quelle f an den Gitterpunkten. Jede Zeile korrespondiert zu einem Gitterpunkt x_{ii}. Die Matrix enthält auf der Diagonalen die 4 als Gewicht von u_{ii}, und in der direkten Nachbarschaft links und rechts die Gewichte -1 der links- und rechtsseitigen Nachbarn u_{i-1j} und u_{i+1j}. Weiterhin gibt es pro Zeile zwei weitere Gewichte -1. Diese sind genau

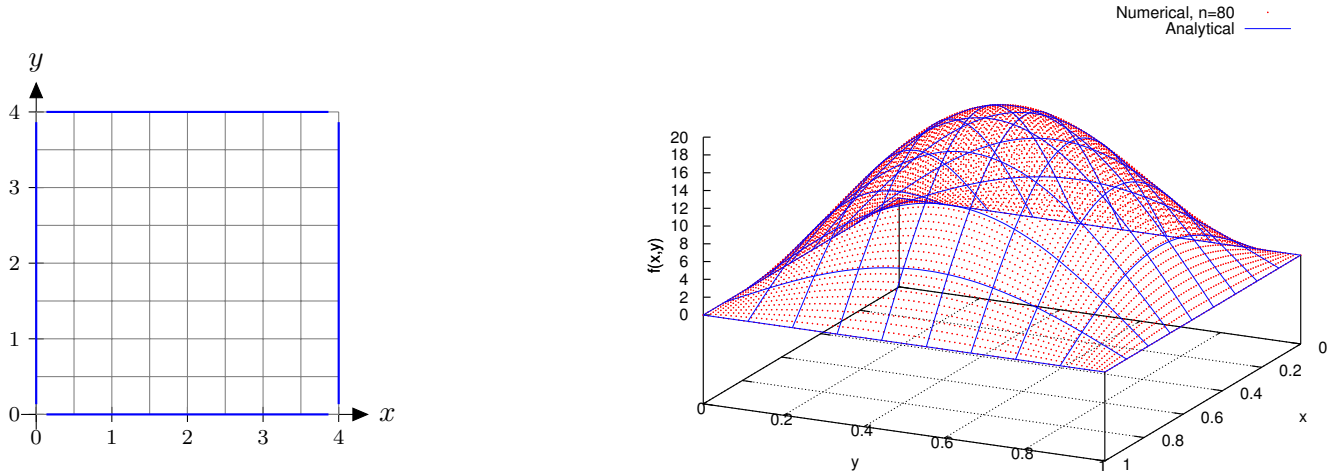

Abbildung 0.5: Links: Gebiet Ω mit Diskretisierung, rechts: Quelle $f(x, y)$.

jene der oberen und unteren Nachbarn u_{ij+1} und u_{ij-1}. Damit ist der Abstand zwischen der -1 rechts neben einer 4 und der -1 daneben gerade n.

An einem Randpunkt muss die zugeordnete Matrix-Zeile durch eine Nullzeile mit einer 1 auf dem Diagonaleintrag, die Lösung u auf 0 und die Rechte Seite auch auf 0 gesetzt werden.

Aufgabe

1. Entwickeln Sie eine MPI-parallele Variante des Jacobi-Verfahrens. Eine mögliche Strategie ist, die Matrix A und die Vektoren in Querstreifen der Höhe (Anzahl Zeilen) α aufzuteilen, jeder Prozessor behandelt einen Querstreifen. Jeder Prozessor erhält in Jacobi-Schritt m eine Kopie der vorhergehenden Lösung $\mathbf{x}^{(m-1)}$, um die neuen Werte x_i seines Streifens zu berechnen. Dann muss in jeder Iteration jeder Prozess die neuen Werte seines Teilbereiches allen anderen Prozessen bekanntmachen, etwa per Multi-Broadcast.

2. Initialisieren Sie die Start-Lösung u^h mit 0.0. Setzen Sie $r = 1.0$ und verwenden Sie eine Toleranz von $\epsilon = 10^{-4}$. Vergessen Sie beim Aufstellen der Matrix und der Initialisierung der Vektorn nicht die Behandlung der Randpunkte! Testen Sie mit Ihrem Code die Konvergenz des Fehlers $\|u - u^h\|_\infty$ zwischen analytischer u und numerischer Lösung u^h an den Gitterpunkten für $n = 4, 8, 16, 32$ und 64. Erstellen Sie einen Plot des Fehlers über der Gitterweite h. Können Sie die Konsistenz-Ordnung erkennen?

3. Messen Sie den Speed-Up im Pool gegenüber der sequentiellen Version ($P = 1$) für verschiedene Problemgrößen $n \leq 32$ und Prozessoranzahlen P.

Hinweise

- Gerne können Sie andere Strategien implementieren, Näheres findes Sie z.B. im Buch *Parallele und verteilte Programmierung* von *Rauber/Rünger*.

- Wenn Sie Schwierigkeiten mit dem Verständnis der Numerik haben, verwenden Sie eine allgemeine Matrix ohne physikalische Anwendung. Damit das Jacobi-Verfahren konvergieren kann, sollte Ihre Matrix A allerdings strikt diagonaldominant sein, d.h. es sollte gelten $\sum_{j=1;j\neq i}^{n} |a_{ij}| < |a_{ii}|$, $\forall i \in \{1, .., n\}$ (Diagonal-Element jeder Zeile betragsmäßig größer als die Betragssumme der Nebendiagonal-Einträge).