

Übungen zur Vorlesung
Paralleles Höchstleistungsrechnen
Dr. S. Lang, J. Pods

Abgabe: 15. Januar 2013, 12 Uhr, per E-mail an jurgis.pods@iwr.uni-heidelberg.de

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 Sie 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 Programm die Konvergenzordnung der Mittelpunktsregel. Erstellen Sie dazu einen doppel-logarithmischen Plot, in dem Sie den Fehler in der Integration über der Intervalllänge 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.gmp.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 π können Sie im Netz besorgen.

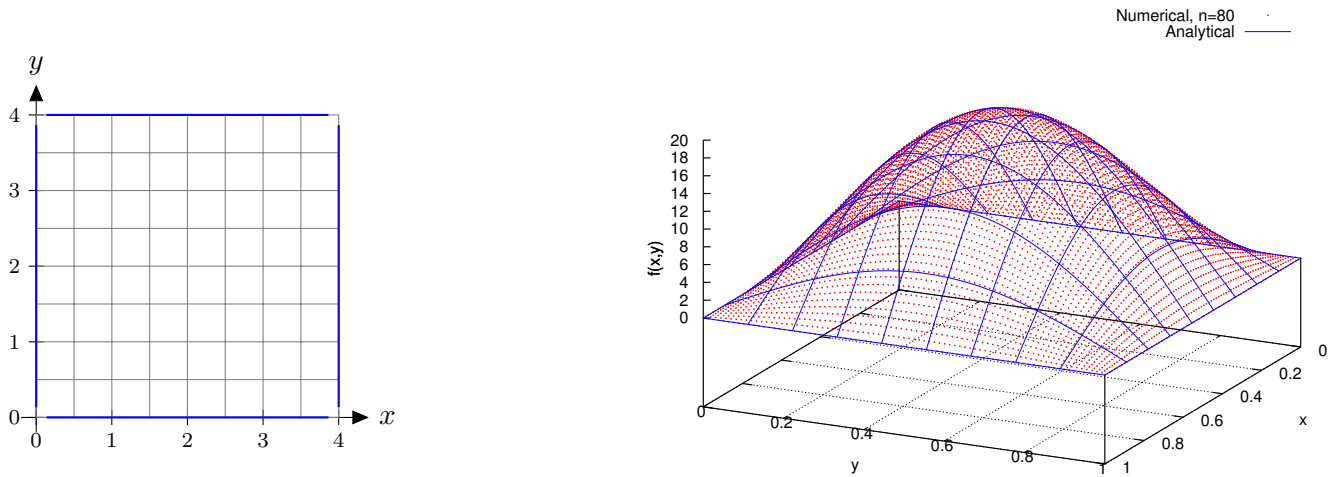


Abbildung 0.5: Links: Gebiet Ω mit Diskretisierung, rechts: Quelle $f(x, y)$.

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 Vektoren 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 finden 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 $\sum_{j=1; j \neq i}^n |a_{ij}| < |a_{ii}|, \forall i \in \{1, \dots, n\}$ (Diagonal-Element jeder Zeile betragsmäßig größer als die Betragssumme der Nebendiagonal-Einträge).