Übungen zur Vorlesung

Wissenschaftliches Rechnen – Paralleles Höchstleistungsrechnen

Prof. Dr. C. Engwer, S. Westerheide
http://wwwmath.uni-muenster.de/num/Vorlesungen/WissenschaftlichesRechnen_SS11/

Abgabe 12. 7. 2011. Abg. der Programmieraufgaben per Email an sebastian.westerheide@uni-muenster.de, schriftliche Abgabe **in Zimmer 120.019**.

Übung 1 Turing-Instabilitäten parallelisieren mit MPI

Auf der Vorlesungsseite finden Sie ein seq. Programm zur Lösung des Anfangswertproblems, wie es sich aus dem Turing-Modell für zwei Chemikalien ergibt:

$$\frac{\partial a}{\partial t} = D_a \Delta a + f(a,b) \qquad \text{auf } \Omega \tag{1}$$
$$\frac{\partial b}{\partial t} = D_b \Delta b + g(a,b) \qquad \text{auf } \Omega \tag{2}$$

mit den aus der Vorlesung bekannten Annahmen bzgl. der Konstanten und Funktionen.

Das Problem der Hydra wird in der Regel in 1D oder 2D betrachtet, ist aber ursprünglich ein dreidimensionales Problem. Es ist schwierig dreidimensionale Musterbildung experimentell zu beobachten, das heißt aber nicht, dass die Prozesse nicht ursprünglich 3D Prozesse sind.

In [Bensagi 2011] wurden chemische Experimente präsentiert, die für die Belousov-Zhabotinsky Reaktion dreidimensionale Musterbildung mit Hilfe eines Tomographen beobachtet wurden. Möchte man solche Experimente im Computer simulieren, stößt man in drei Dimensionen schnell an die Leistungsgrenze des Computers, daher werden Parallelisierungsstrategien interessant.

Das seq. Programm implementiert die im [Bensagi 2011] vorgestellten Reaktionsterme

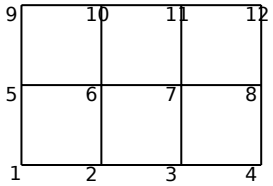
$$f(a,b) = 1/\epsilon_0(w_0 a + w_1 b - a^2) \tag{3}$$
$$g(a,b) = w_0 a - b \tag{4}$$

mit

$$w_0 = (1.0 - m \cdot b)/(1.0 - mb + \epsilon_1)$$
$$w_1 = f(q-a)/(q+a)\ .$$

Das seq. Programm unterstützt Simulationen in 2D und 3D, wir werden uns jedoch im folgenden auf Simulation in 2D beschränken.

1. Machen Sie sich mit dem seq. Programm vertraut.
   - Sie können im Hauptprogramm die Gitterauflösung mit dem Parameter D einstellen. Je größer D ist, desdo mehr Zellen werden in jeder Raumrichtung verwendet.
   - Wie verhält sich die Lösung, wenn Sie auf feineren Gittern rechnen.
   - Wie fein müssen Sie rechnen, damit die Gitterweite keinen Einfluss auf die Lösung mehr hat? (*Hinweis:* Eventuell möchten Sie zur Beantwortung dieser Frage erst das Programm parallelisieren)
2. Parallelisieren Sie die Berechnung des Turing Problems, durch parallele Datenaufteilung (Hinweise hierzu werden auch nochmal in der Vorlesung am 30. 6. 2011 kommen).
   - Das Programm verwendet einen Splitting-Ansatz um Orts- und Zeit-Diskretisierung zu entkoppeln.
   - Die Ortsdiskretisierung verwendet ein strukturiertes Gitter und eine Finite-Differenzen Diskretisierung. Die Knoten des Gitters werden lexikographisch durchnummeriert und zu jedem Knoten wird die aktuelle Konzentration von a und b in einem Vektor gespeichert.



2D Gitter mit 3×4 Knoten, sowie deren Nummerierung.

Angenommen wir haben ein Gitter mit $N \times M$ Gitterpunkten, so hat ein Knoten i im Inneren des Gebiets die Nachbarn $i-1$ und $i+1$ in x-Richtung, sowie $i+N$ und $i-N$ in y-Richtung.

Die 2. Ableitungen des Laplace-Operators werden durch Differenzen-Quotienten approximiert. Für die 2. Ableitung im Punkt i in x-Richtung ergibt sich somit

$$\partial a_i = \frac{a_{i-1} + a_{i+1} - 2a_i}{h}. \tag{5}$$

Wobei h die Gitterweite beschreibt. Wenn $\mathcal{N}_i$ die Menge der Nachbarknoten von i ist, kann man den Laplace-Operator als

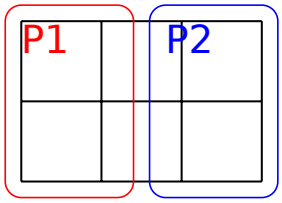
$$\Delta a_i = \sum_{j \in \mathcal{N}_i} (a_j - a_i) \tag{6}$$

berechnen. Diese Formulierung berücksichtigt auch die Neumann-Randbedingungen korrekt (am Rand gibt es weniger Nachbarknoten).

- Die Zeitableitung wird durch des explizite Eulerverfahren berechnet, d.h. der Laplace Operator und die Reaktionsterme werden bzgl. des alten Zeitschrittes ausgerechnet und liefern ein Update auf den neuen Zeitschritt. Die alles geschieht in der Funktion `compute_update`, welche wiederum für jede Zelle `compute_local_update` aufruft.

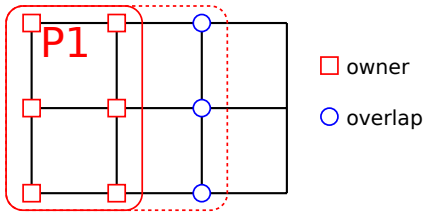
  Das explizite Eulerverfahren ist nicht uneingeschränkt stabil. Es muss die CFL-Bedingung eingehalten werden, welche eine Beschränkung des Zeitschritts mit sich bringt. Hierzu wird in der Funktion `compute_update` basierend auf den aktuellen Flüssen und Reaktionsraten.

Zur Parallelisierung müssen jetzt die Daten parallel verteilt werden. Die geschieht, indem das Gebiet in Teilgebiete zerlegt wird, welche parallel bearbeitet werden. Dazu wird das Gebiet in Rechtecke zerteilt, die möglichst gleich viele Knoten enthalten und möglichst „rund" sind, d.h. die Kantenlängen jedes Rechtecks sollen möglichst gleich groß sein. Hierzu muss die Klasse `structured_grid` (in `grid.hh`) erweitert werden.



Parallele Aufteilung der Gitterknoten auf zwei Prozesse P1, P2.

Da zur Anwendung des Laplace-Operators benötigen wir die Nachbarn aller Knoten unseres Patches (owner-Knoten). Daher vergrößern wir unseren Patch in alle Richtungen um eine Reihe von Knoten, diese nennen wir overlap-Knoten. Alle diese Knoten werden nun durchnummeriert und beschreiben unsere lokale Daten.



Vergrößerung des Patches von P1 & Identifizierung der owner-, sowie overlap-Knoten.

- `compute_update` muss angepasst werden, so dass nur für ower-Knoten ein Update berechnet wird. *Hinweis*: hierzu müssen zusätzliche Information vom Gitter bereitgestellt werden.
- Die Berechnung der Zeitschrittschranke erfordert die Bestimmung des global kleinsten zulässigen Zeitschritts, hierzu ist eine alle-an-alle Kommunikation erforderlich.
- Da kein Update für overlap-Knoten berechnet wird, steht nach der Aktualisierung des Lösungsvektors (in `main`) für diese der falsche Wert im Vektor. Um den richtigen Wert zu erhalten müssen mit den Nachbarprozessen die Daten ausgetauscht werden. Dazu ist es am besten das Gitter um eine weitere Methode zur Kommunikation zu erweitern. Es sollen so wenig wie möglich Daten verschickt werden, d.h. nur die Daten für den Overlap-Bereich.

- Das Schreiben der Visualisierungdaten sollte von einem Prozess durchgeführt werden, um Koordinationsprobleme beim Zugriff auf die Datei zu vermeiden.

  Die einfachste Möglichkeit ist es, dass alle Prozesse ein Kopie des kompletten Vektors an Prozess 0 schicken, welcher dann immer die Daten dieses Patches in die Datei schreibt. Es ist zu beachten, dass jetzt die Daten nicht mehr einfach hinten angehängt werden dürfen, sondern, dass man an die richtige Stelle innerhalb der Datei springen muss. Dies kann man mit dem Befehl `fseek` machen.

3. Testen sie ihr paralleles Programm und vergleichen sie die Laufzeit mit dem seq. Programm.

12 Punkte

Übung 2 Bonusaufgabe

Das seq. Programm funktioniert sowohl für 2D als auch 3D Probleme.

- Erweitern Sie ihre Parallelisierung, so dass auch dies für 3D Berechnungen funktioniert.
- Führen Sie Simulationen für die Parameter des 2D Problems aus Aufgabe 1 durch.
- Beginnen Sie wieder mit Rechnungen auf relativ groben Gittern. Wie skaliert die Rechenzeit für feinere Gitter?
- *Hinweis:* Eventuell müssen Sie den Endzeitpunkt anpassen, um den stationären Zustand zu erreichen.

6 Punkte