Goethe-Center for Scientific Computing (G-CSC)
Goethe-Universität Frankfurt am Main

Modellierung und Simulation II

(Praktikum SIM2, SoSe 2017)

M. Breit, Dr. A. Vogel

Blatt 7 (Abgabe: Mo., 03.07.2017, 10h)

Auf diesem Blatt implementieren Sie erste Strukturen zur Parallelisierung. Dafür benötigen Sie eine funktionsfähige MPI-Bibliothek, etwa Open MPI.

Aufgabe 1 (Paralleles strukturiertes Gitter, 4 Punkte)
Das bisher nur auf einem Prozess liegende Gitter soll nun verteilt werden. Um die Sache möglichst einfach zu halten, lassen wir nur solche Kombinationen von Gittern und Prozessorzahlen zu, die eine Verteilung von gleichförmigen Teilgittern auf die einzelnen Prozesse erlauben, d.h., wir lassen für ein Gitter der Raumdimension d nur Prozessanzahlen $P = n^d$ ($n \in \mathbb{N}$) derart zu, dass das strukturierte Gitter $n_1 \times n_2 \times \cdots \times n_d$ Elemente hat und jedes n_i ein Vielfaches von n ist. Dann bekommt jeder Prozess ein Teilgitter mit $\frac{n_1}{n} \times \frac{n_2}{n} \times \cdots \times \frac{n_d}{n}$ Elementen. Erweitern Sie den Konstruktor Ihrer Implementierung von `StructuredGrid` um eine entsprechende Prüfung und schneiden Sie die Gitter-Abgrenzungen für jeden Prozess passend zu. Ordnen Sie die Teilgitter genauso an wie die Knoten-Indices (also lexikographisch).

Aufgabe 2 (Layouts und Interfaces, 8 Punkte)
Als nächstes sollen die (horizontalen) Interfaces erzeugt werden:

(a) Implementieren Sie die Klasse `Layout`, die eine Kollektion von Interfaces (etwa: alle Master-Interfaces eines Prozesses) verwaltet,

(b) außerdem die Klasse `GridLayoutManager`, eine Singleton-Klasse, welche die Master- und Slave-Layouts für jeden Prozess hält und für alle Objekte, die diese benötigen, zugänglich macht.

(c) Implementieren Sie die Methode `StructuredGrid::store_interfaces()`, welche die Master- und Slave-Layouts im `GridLayoutManager` berechnet. Aufgrund der gewählten Einschränkungen an Gitter und Prozessor ist dies ohne jegliche Kommunikation möglich. Dabei soll unter allen Prozessen, die einen Knoten halten, immer derjenige mit dem niedrigsten Rang Master sein, die anderen Slave. Achten Sie darauf, dass die Indices, die in den Interfaces gespeichert werden, geordnet sind.

Aufgabe 3 (Parallele Vektoren, 8 Punkte)
Neben dem Gitter muss auch die Algebra parallelisiert werden. Die Gitter-Vektoren auf jedem Prozess enthalten nur noch die Werte zu Grid-Indices, die der Prozess besitzt. Da manche Vertices auf mehr als einem Prozess existieren, ist es notwendig, den parallelen Speicherzustand (consistent, additive, unique) des Vektors in den Interfaces zu kennen.

(a) Implementieren Sie die nützlichen MPI-Wrapper-Routinen `all_reduce` und `distribute_data` in »mpi_wrapper.h«.

(b) Parallelisieren Sie Ihre Vektor-Klasse `Vector`, indem Sie die zusätzlichen Methoden in »vector.h« implementieren. Beachten Sie auch, dass Sie den parallelen Speicherzustand bei Vektor-Operationen (+, -, *) und in Konstruktoren überprüfen bzw. setzen müssen.

(c) Überarbeiten Sie insb. Ihre Implementierung von `Vector::norm()`, die nun auf einem parallelen Vektor stattfinden muss.

Abgabe: Senden Sie Ihren Code sowie sonstige Antworten als Text, PDF oder Scans bitte per E-Mail an `practical.sim2@gcsc.uni-frankfurt.de`. An diese Adresse können Sie sich auch bei Fragen zu den Aufgaben wenden.