1 Parallele Algorithmen – Grundlagen

Parallele Algorithmen – Grundlagen

Wir unterscheiden folgende drei Schritte im Design paralleler Algorithmen:

Dekomposition eines Problems in unabhängige Teilaufgaben. *Dies identifiziert die maximal mögliche Parallelität.*

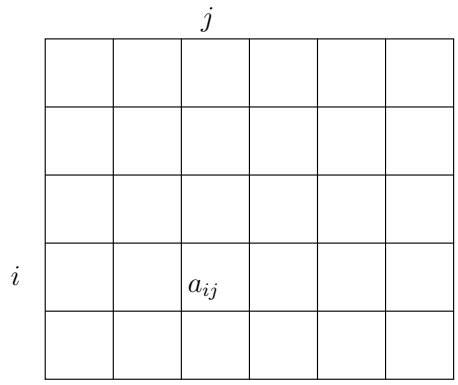
Kontrolle der Granularität um Rechenaufwand und Kommunikation auszubalancieren.

Abbilden der Prozesse auf Prozessoren um die logische Kommunikationsstruktur möglichst gut auf die Maschinenstruktur abzustimmen.

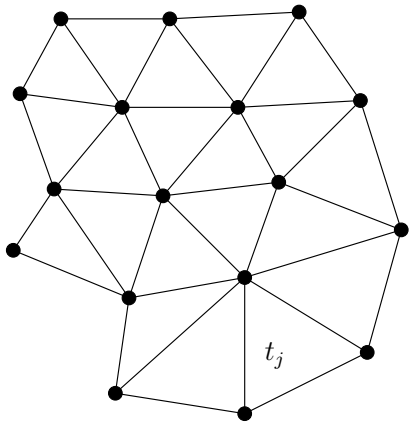
1.1 Zerlegung

Viele Algorithmen sind an spezielle Datenstrukturen gebunden. Gewisse Operationen müssen für jedes Datenobjekt ausgeführt werden.

Beispiel: Matrix-Addition C = A + B *oder* Triangulation



Matrix



Triangulation

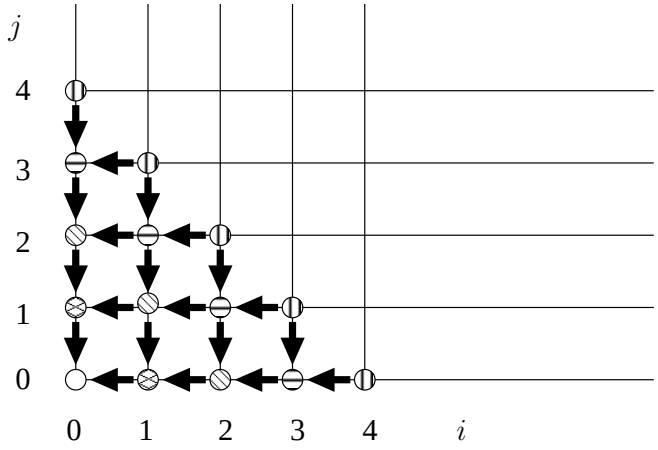
Datenabhängigkeiten

Oft können Operationen nicht für alle Datenobjekte gleichzeitig ausgeführt werden.

Beispiel: Gauß-Seidel/SOR-Iteration mit lexikographischer Nummerierung.

Berechnungen werden auf einem Gitter durchgeführt, wobei die Berechnung am Punkt (i, j) vom Ergebnis der Punkte $(i-1, j)$ und $(i, j-1)$ abhängt. Gitterpunkt $(0, 0)$ hat keine Abhängigkeiten. Nur Punkte auf der Diagonalen $i + j = const$ können gleichzeitig berechnet werden.

Datenabhängigkeiten machen die Datenzerlegung wesentlich komplizierter.

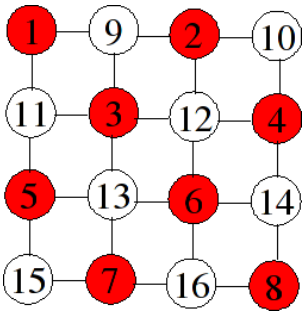


Erhöhen der möglichen Parallelität

Manche Algorithmen kann man modifizieren, um eine höhere Datenunabhängigkeit zu erhalten.

Durch eine andere Nummerierung der Unbekannten, kann der Parallelisierungsgrad des Gauß-Seidel/SOR Verfahrens erhöht werden:

Jeder Punkt im Gebiet bekommt eine Farbe zugeordnet, so dass zwei Nachbarn niemals die gleiche Farbe haben. Für strukturierte Gitter sind zwei Farben ausreichend (red/black). Die Unbekannten einer Farbe werden zuerst nummeriert, dann die der nächsten Farbe ...

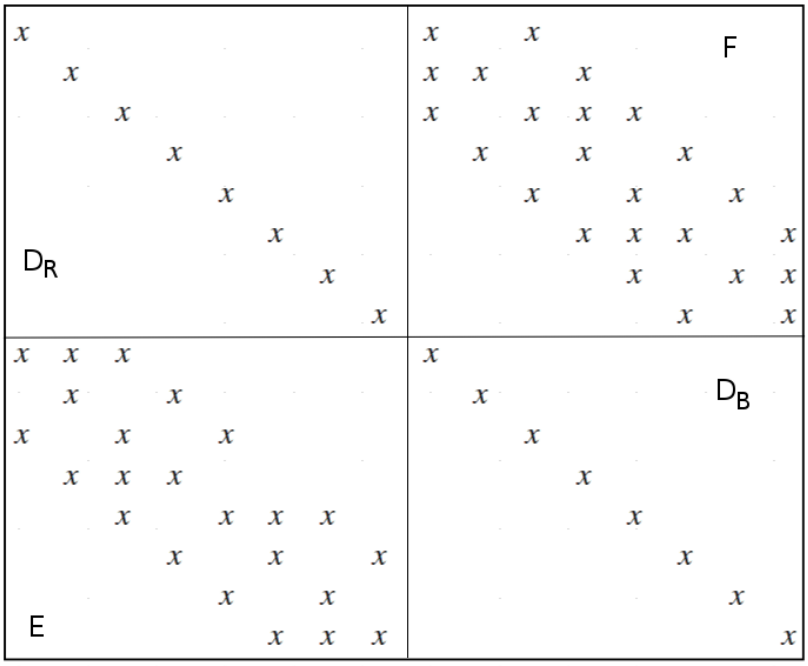


Red-Black-Sortierung

Die Gleichungen aller Unbekannten mit gleicher Farbe sind unabhängig von einander. Für strukturierte Gitter, erhalten wir eine Matrix der Form

$$A = \begin{pmatrix} D_R & F \\ E & D_B \end{pmatrix}$$

Obwohl eine solche Matrixumformung keinen Einfluss auf die Konvergenz von Krylov-Raum-Verfahren (z.B. CG oder GMRes) haben, beeinflusst sie die Konvergenzrate von Relaxationsverfahren.

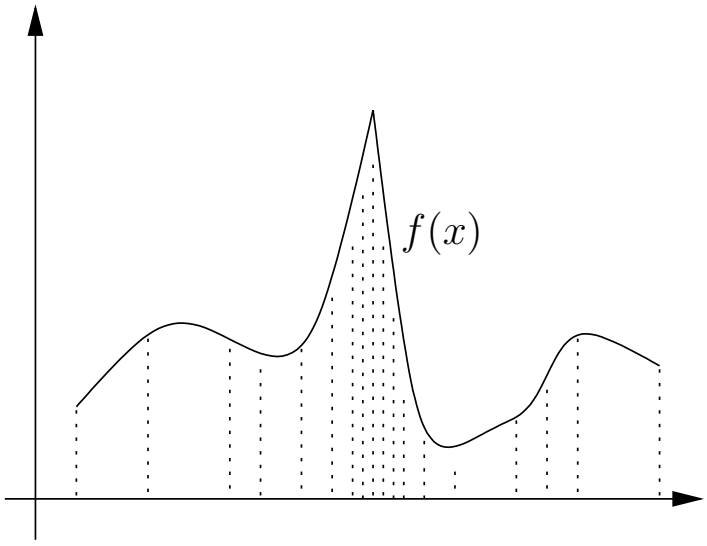


Irreguläre Probleme

Bei manchen Problemen kann man die Zerlegung nicht a priori festlegen.

Beispiel: Berechnung des Integrals einer Funktion $f(x)$ durch adaptive Quadratur.

Die Wahl der Intervalle hängt von der Funktion f ab und ergibt sich erst während der Berechnung durch Auswertung von Fehlerschätzern.



1.2 Agglomeration

Agglomeration und Granularität

Die *Zerlegung* zeigt die *maximale Parallelität* auf. Meist ist es nicht sinnvoll, diese wirklich zu nutzen ($\rightarrow$ ein Datenobjekt pro Prozess), da der Kommunikationsaufwand dann zu groß wird.

Daher: Man ordnet einem Prozess mehrere Teilaufgaben zu und fasst die Kommunikation für all Teilaufgaben in möglichst wenigen Nachrichten zusammen.

Dies nennt man *Agglomeration.*

Die Anzahl der zu sendenden Nachrichten wird reduziert.

Als *Granularität* eines parallelen Algorithmus bezeichnet man das Verhältnis

$$\text{granularity} = \frac{\text{number of messages}}{\text{computation time}}$$

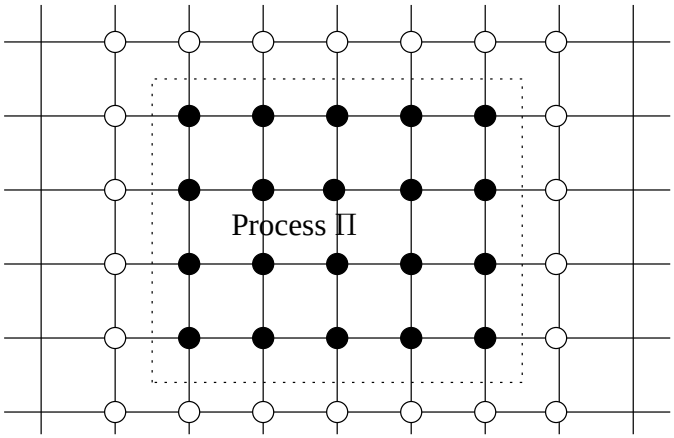
Agglomeration reduziert die Granularität.

Beispiel: Gitterbasierte Berechnungen

Jedem Prozess wird eine Menge von Gitterpunkten zugewiesen.

In iterativen Verfahren wird oft der Wert an jedem Knoten, sowie die seiner Nachbarn benötigt. Wenn keine Datenabhängigkeiten (siehe Gauß–Seidel) bestehen, können alle Modifikationen gleichzeitig durchgeführt werden.

Beispiel 2D Gitter: Ein Prozess mit N Gitterpunkten muss $O(N)$ Operationen durchführen. Mit der der gewählten Partitionierung muss er lediglich $4\sqrt{N}$ Randwerte kommunizieren. Das Verhältnis von Kommunikations- und Berechnungsaufwand ist daher $O(N^{-1/2})$ und wird beliebig klein für wachsendes N. Dies nennt man *Oberfläche-zu-Volumen-Effekt.*



1.3 Abbilden der Prozesse auf Prozessoren

Abbilden der Prozesse auf Prozessoren

Die Menge Π der Prozesse bildet einen ungerichteten Graph $G_\Pi = (\Pi, K)$. Zwei Prozesse sind verbundet, wenn sie kommunizieren.

Die Menge P der Prozessoren bildet mit dem Kommunikationsnetzwerk (Hypercube, Feld, ...) ebenfalls einen Graph $G_P = (P, N)$.

Annahme $|\Pi| = |P|$.

Welcher Prozess ist von welchem Prozessor auszuführen?

Im allgemeinen soll die Abbildung so aussehen, dass die kommunizierende Prozesse auf möglichst (nahe) benachbarte Prozessoren abgebildet werden. *Dieses Optimierungsproblem bezeichnet man als Graphabbildungsproblem und ist leider $\mathcal{NP}$–vollständig.* In *cut-through* Netzwerken ist die Übertragungszeit fast entfernungsunabhängig, daher ist das Problem der Prozessplatzierung heute weniger relevant. Wenn jedoch viele Prozesse gleichzeitig miteinander kommunizieren (was aber durchaus häufig vorkommt!), ist eine gute Positionierung immernoch wichtig.

1.4 Lastverteilung

Lastverteilung: Statische Aufteilung

Bin Packing Am Anfang sind alle Prozessoren leer. Knoten werden nacheinander auf den Prozessor gepackt, der am wenigsten Arbeit hat. *Funktioniert auch dynamisch, wenn während des Rechenvorganges neue Arbeit entsteht.*

Recursive Bisection Es wird die zusätzliche Annahme gemacht, daß die Knoten eine Position im Raum besitzen. Der Raum wird jetzt so zerteilt, daß in den Teilen etwa gleich viel Arbeit besteht. Dieses Vorgehen wird dann rekursiv auf die entstandenen Teilräume angewandt.