Inhaltsverzeichnis

1. Parallele Systeme

Üblicherweise und auch als Default-Einstellung werden alle Operationen im Rechner sequentiell bearbeitet. Ein ausführendes Programm bekommt die Daten und erzeugt genau einen Thread mit einem Satz von Instruktionen (Anweisungen) für die Datenverarbeitung. Diese Art der Systeme werden als SISD (Single Instruction Single Data) bezeichnet. Die Instruktionen in einem Thread werden bei dieser Systemarchitektur sequentiell nacheinander durchgeführt. Jede von den Instruktionen dabei bekommt und bearbeitet eine minimalen Datenmenge. Die Performanz der durchgeführten Operationen ist niedrig. Die Leistungserhöhung ist heutzutage eine von den wichtigsten Aufgaben der Informatik.

Die Performanz durchgeführten Operationen kann durch die parallele Datenverarbeitung mithilfe mehrere gleichzeitig ausgeführten Threads deutlich erhöht werden.

Parallele Systeme sind Rechner- und Softwarearchitekturen, die eine parallele Datenverarbeitung mit einem möglichen Konzept, oder auch mit einer Kombination von Frameworks, auf einem Rechner oder in einem Netzwerk ermöglichen. Die Hauptidee des Parallelismus basiert sich auf dem Prinzip „Teile und Herrsche“. Die Mehrheit der Bearbeitungsaufgaben können in kleine Unteroperationen zerlegt werden, die können gleichzeitig unabhängig voneinander ausgeführt werden. Die Koordination der durchgeführten Unteroperationen kann in einigen Fällen eine entschiedene Rolle spielen und muss unbedingt berücksichtigt werden.

Alle Formen des Parallelismus können in die folgenden Kategorien unterteilt werden:

- Bit-Ebene

  Die Forme des Parallelismus basiert sich auf die Erweiterung von Datenwort (Binärwort). Je größer ist das Datenwort desto, weniger Operationen müssen durchgeführt werden, um eine Aufgabe zu erfüllen. Um zwei 16-Bit-Zahlen zusammen zu addieren, muss ein 8-Bit-Prozessor 3 Instruktionen ausführen. Ein 16-Bit-Prozessor erfüllt die Aufgabe mit einer Instruktion.

- Instruktion-Ebene

  Jedes Programm ist ein Satz von Instruktionen. Die Reihenfolge der durchgeführten Instruktionen kann ohne den Einfluss auf die Ergebnisse der Ausführung vom ganzen Programm geändert werden aber für die Performanz entscheidend sein.

- Daten-Ebene

Bei dieser Methode werden alle Daten gleichzeitig bearbeitet werden. Die Daten werden unter Prozessen auf einem Rechner oder unter Rechner in einem Netzwerk verteilt werden. Für die gleichmäßige Verteilung sorgt sich die spezielle Software.

- Aufgabe-Ebene

  Die Forme des Parallelismus basiert sich auf die Zerlegung einer Bearbeitungsaufgabe in die die kleine voneinander unabhängige Unteroperationen, die mit verschiedenen Prozessen oder Rechner gleichzeitig ausgeführt werden.

In weiteren Abschnitten (1.1 und 1.2) werden zwei Programmierungskonzepte für die Parallelisierung, nämlich Open MP und Open MPI, vorgestellt.

1.1 OpenMP

OpenMP steht für Open Multi-Processing. Das ist ein offenes Konzept für die parallele Programmierung für C/C++ und Fortan. Das Framework beschreibt einen Satz von Direktiven für den Compiler. Dieses ermöglicht Programme nach dem Prinzip „Shared Memory“ zu parallelisieren.

OpenMP realisiert die Parallelisierung mit Hilfe von Multi-Threading. Bei dem Konzept wird ein Master-Thread erzeugt, der in seiner Reihe die Slave-Threads generiert und verwaltet. Die Bearbeitungsaufgaben werden unter den Threads verteilt. Es wurde festgelegt, dass die Threads gleichzeitig auf einem Rechner ausgeführt werden. Die wichtige Bedingung ist dabei, dass die Anzahl von Rechner-Prozessoren (Kernen) der Anzahl der erzeugten Threads entspricht oder weniger sein muss (Ein Rechner-Prozessor – ein Thread).

Die parallelisierende Programmierungaufgaben werden mit Hilfe von speziellen Direktiven von Preprozessor beschrieben. Die Programmierungssprache – pragm.

Die Hauptdirektiven sind:

- Direktive für das Multi-Threading – `parallel`
  Für die Thread-Genenierung ist zuständig
- Direktiven für die Arbeitsverteilung unter Threads – `DO/for` und `section`
- Direktiven für den Datenzugriff unter Threads– `shared` und `private`
- Direktiven für die Synchronisierung der Arbeit unter Threads – `critical`, `atomic` und `barrier`
- Direktive für die Identifizierung eines bestimmten Thread – `omp_get_thread_num`

1.2 Open MPI

MPI steht für Message Passing Interface. Das ist ein offener Standard für die Kommunikation zwischen Blocken in einem parallelisierten Programm. Das Konzept ist beabsichtigt für die Unterstützung der Arbeit von parallel ausgeführten Prozessen. Die Funktionsweise basiert sich auf das Nachrichtenversenden zwischen den Prozessen. Das Framework ermöglicht die Opitimierung auf einem Rechner sowie in einem Netzwerk. Das Prinzip der Parallelisierung ist „shared memory“ und „peer-to-peer“.

Alle Prozessen werden in die Gruppen unterteilt. Diese werden Kommunikatoren genannt. Ein Prozess kann gleichzeitig zu mehreren Kommunikatoren zugeordnet werden aber wird als Defaul-Einstellung in dem Kommunikator MPI_COMM_WORLD mit einem Rank als Identifikator

registriert. Der Prozess mit dem Rank 0 ist ein Master-Prozess. Dieser verwaltet die Logik der Anwendung und verwaltet alle anderen sogenannten Slave-Prozessen, die durch die Eingabe erzeugt werden.

Ein minimales Programm mit MPI kann mit dem folgenden Listing mit den Kommentaren realisiert:

```
1. int main(int argc, char **argv) {
2. int rank, size;
3. //Initialisierung
4. MPI_Init(&amp;argc, &amp;argv);
5. //Anzahl der erzeugten Prozessen
6. MPI_Comm_size(MPI_COMM_WORLD, &amp;size);
7. //Die Prozesse werden in Kommunikator registriert
8. //Jedem Prozess wird einen Rank zugeordnet
9. MPI_Comm_rank(MPI_COMM_WORLD, &amp;rank);
10.//Wenn der Master-Prozess startet wird „Hello World main process“ in
11.//die Konsole ausgegeben
12.if(rank == 0) {
13.printf(&quot;Hello world, I<code>m main process! There are %d processes
14.in my comm.\n&quot;, size);
15.   }
16.//Wenn der Slave-Prozess startet wird „Hello World“ in die Konsole
17.//ausgegeben
18.else {
19.printf(&quot;Hello world! I</code>m %d process in comm.\n&quot;, rank);
20.   }
21.//Löscht alle Prozesse
22.MPI_Finalize();
23.return 0;
24.}
```

Für die Datenverteilung zwischen den Slave-Prozessen wurden auch bei MPI eigene Basis-Datentypen definiert. Darunter sind MPI_INT, MPI_CHAR, MPI_BYTE und andere.

Der Hauptvorteil von MPI ist das, dass das Framework ist heterogen und dadurch komplett plattformunabhängig. Die parallelisierten mithilfe MPI Programme können problemlos von Cluster zu Cluster, ohne den Code anzupassen, übertragen werden. Um die Performanz der ausgeführten Programmen noch mehr zu erhöhen, müssen aber die Besonderheiten des Clusters berücksichtigt werden.

2. Digitale Bildverarbeitung

Unter der digitalen Bildverarbeitung versteht man die Manipulationen mit Signalen (Bildinformationen) in jeder Art von digitalen Bildern. Als Eingangsbilder können beliebige digitale Bilder sein.

Bildverarbeitung kann wie für die Acquisition eines neuen Bildes (z.B. Bild einer besseren Qualität oder Verarbeitung eines Bildes für die Polygraphie), sowie für die Extraktion von den neuen Informationen (z.B Texterkennung oder Informationen über die Anzahl und Art der Zellen bei einer mikroskopischen Untersuchung) dienen.

Ein digitales Bild kann mithilfe einer Bildfunktion mit zwei Variablen dargestellt werden ***I(x,y)***, wo ***I*** ist die Intensität (Helligkeit) in einem bestimmten Punkt mit den Koordinaten ***(x,y)***. Manchmal muss nicht das ganzes Bild, sondern ein Teil oder ein Abschnitt bearbeitet werden. In dem Fall wird nur ein Bereich des Bildes, sogenannte ***ROI (Region of Interest),*** bei der Verarbeitung

berücksichtigt werden.

Digitales Bild kann in zweidimensionalen Raum als eine Matrix ***A(m.n)*** dargestellt werden, wo ***m*** ist Nummer der Zeile im Bild und ***n*** ist die Nummer der Spalte. Das Element auf dem Schnittpunkt von m und n wird als Pixel bezeichnet. Die Intensität eines Pixels kann ganzzahlig (von 0 bis 255) oder mit einer Gleitkommazahl (von 0 bis 1) dargestellt werden.

Alle digitale Bilder können in zwei Hauptkategorien unterteilt werden:

- Farbbilder (RGB, CMYK und andere)
  Darunter versteht man ein Bild, beim ein Intensität eines Pixels durch mehrere Farbkanalen (oder auch anderen Parametern) dargestellt werden. Die Intensität eines Pixels von einem RGB-Bild wird mithilfe drei Kanalen: Rot, Grün und Blau repräsentiert(Abb.1).

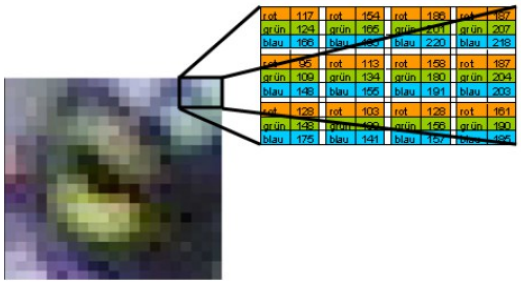


Abbildung 1: Farbbild. Farbraum: RGB - ein Pixel ganz wird nah betrachtet

- Grauwertbilder
  Die Intensität eines Pixels von einem Grauwertbild wird nur mit einem Wert im Bereich von 0 bis 255 repräsentiert. Grauwertbild ist ein Monokanalesbild.

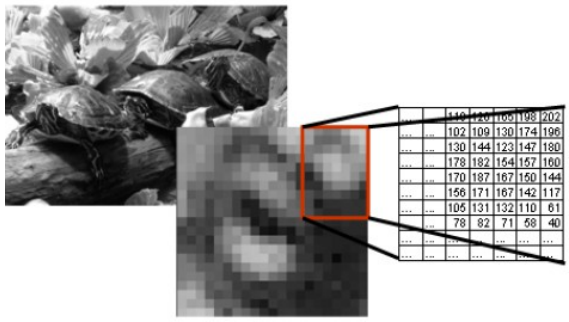


Abbildung 2: Grauwertbild. Pixel wird ganz nah betrachtet

Bildverarbeitung hat mehrere Ziele:

- Verbesserung der Qualität (z.B. Rausch löschen)
- Merkmalextrahierung (Gewinnung der nötigen Informationen für die weitere Analyse)
- Objekterkennung
- Filterung und andere.

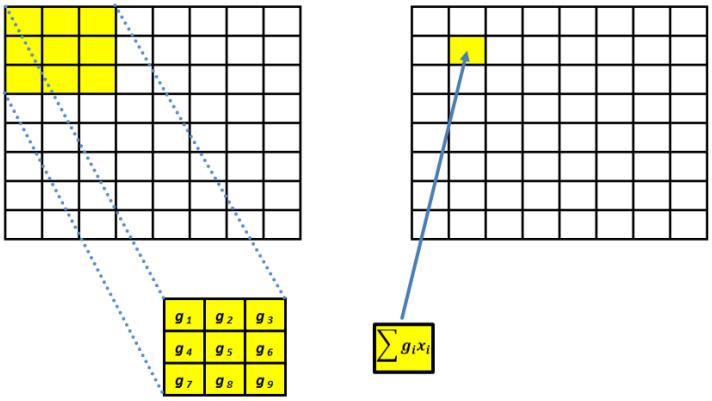
Ein vollständiger Bildverarbeitungsprozess hat die nacheinander folgenden Schritten:
Bildaufnahme => Diskretisierung => Verbesserung => Segmentierung => Merkmalsextraktion => Klassifikation => Auswertung der Ergebnisse

In weiteren Abschnitten (2.1 - 2.4) werden einige Methoden der Bildverarbeitung beschrieben.

2.1 Filterung – Tiefpassfilter

Filterung ist eine Methode der Bildverarbeitung, die für die Verbesserung der Bildqualität und für die Verdeutlichung der gewünschten Merkmalen dient. Beim der Anwendung eines Filters wird das Originalbild so verändert, dass die gewünschten Merkmale mehr sichtbar werden. In der Regel Filterung wird für die Glättung(z.B. technische Störungen löschen) oder Verschärfung (Betonung die Kanten) eines Bildes verwendet.

Die Filtermatrix besteht aus einem rechteckigen oder quadratischen Bildausschnitt (Kernel, Filter), der auf die Bildkoordinate (x, y) zentriert und dann von Pixel zu Pixel verschoben wird.



2.2 Thresholding

2.3 Openning &Closing

2.4 Hochpassfilter

3. Hough Transformation

4.Implementierung

5.Benchmarks

6. Ausblick