TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Technische Informatik, Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Übertragung eines an Mapreduce orientierten k-Means Algorithmus auf einen FPGA-Hardwarebeschleuniger für den Cloud-Einsatz

Studienarbeit

Martin Knöfel

Dresden, 12. Oktober 2017

Ziele der Arbeit

- Literaturstudium zum Map-Reduce, Vivado High-Level-Synthese (VHLS) und k-Means Algorithmus
- Entwurf und Design des C++ Sourcecodes für VHLS
- Spezifizierung der Synthese mit VHLS Direktiven
- Implementierung und Auswertung

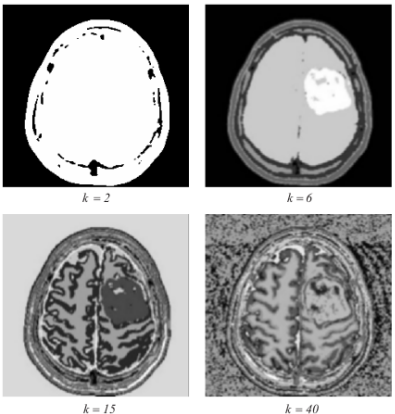
Gliederung des Vortrags

1. Motivation
2. Literatur
3. Entwurfsprozess (HLS)
4. Auswertung
5. Zusammenfassung und Ausblick

k-Means Clustering

- übersichtliches Clusterverfahren
- großer Anwendungsbereich:
  - *Bildverarbeitung*
  - *Medizin*
  - *Statistik*
  - *Machine Learning*
- hoher/steigender Bedarf an Datensegmentierung

k-Means Beispiel



Cloud-Einsatz

- geringe Einstiegskosten für kleine Unternehmen
- spezielle Hardware (GPUs, FPGAs, ...)
- Skalierbarkeit der Ressourcen
- effiziente Nutzung der Hardware (zeitlich und räumlich)

High-Level-Synthese

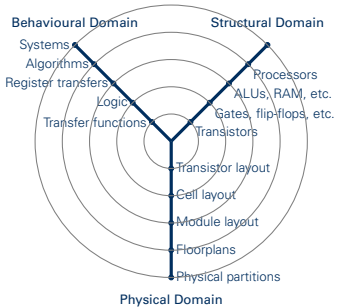
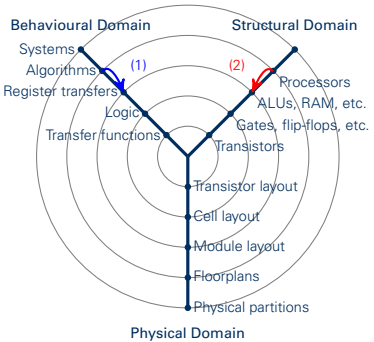


Abbildung: Gajski-Kuhn Y-Diagramm [1]

[1] Hemani, 2004: "Charting the EDA roadmap"

Vivado High-Level-Synthese



- (1) generiert VHDL/Verilog Code aus C/C++ Programm
- (2) Spezifikation der Hardwarestruktur über Direktiven(Pragmas)

IDEE:

- grundlegenden Struktur des k-Means Algorithmus
- Anpassung an den Bedarfsfall (Anwendung/Cloud)
- effiziente Implementierung durch die High-Level-Synthese

Map-Reduce

- Programmiermodell von Google [2]
- automatisierte Aufteilung von Daten
- zwei Schritte:

  MAP: Verteilung der Daten auf die Verarbeitungsknoten
  REDUCE: Zusammenfassen der Teilergebnisse

- k-Means → Abstandsberechnung, Zuordnung [3]

[2] Dean u. a., 2008: "MapReduce: simplified data processing on large clusters"

[3] Choi u. a., 2014: "Map-reduce processing of k-means algorithm with FPGA-accelerated computer cluster"

k-Means Allgemein

- Aufteilung von Daten in k-Cluster
- mittels Abstandsberechnung und Zuordnungsmethode
- terminierend und konvergiert zu einem lokalen Optimum [4]
- Standard-Algorithmus: Lloyd-Algorithmus

[4] Selim u. a., 1984: "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality"

k-Means Abstandsberechnung

quadratische euklidische Distanz (Lloyd-Algorithmus):

$$d = \sum_{j=1}^{k} (a_j - b_j)^2$$

Manhatten-Distanz:

$$d = \sum_{j=1}^{k} |a_j - b_j|$$

Kosinus-Distanz:

$$d_{st} = 1 - \frac{x_s x_t'}{\sqrt{(x_s x_s')(x_t x_t')}}$$

Korrelations-Distanz:

$$d_{st} = 1 - \frac{(x_s - \overline{x_s})(x_t - \overline{x_t})}{\sqrt{(x_s - \overline{x_s})(x_s - \overline{x_s})'}\sqrt{(x_t - \overline{x_t})(x_t - \overline{x_t})'}}$$

5

[5] Bora u. a., 2014: "Effect of different distance measures on the performance of K-means algorithm: an experimental study in Matlab"

k-Means Pseudocode

```
while(Zentren != alte Zentren)
{
    for (alle Datenpunkte)
    {
        for (alle Zentren)
        {
            berechneAbstand(Datensatz, Zentrum);
        }
        kleinsten_Abstand();
    }
    berechne_Zentren_neu();
}
```

Verwandte Arbeiten

- Ein- und Ausgabe der Daten ist ein Flaschenhals
- Geschwindigkeit abhängig von der Menge der Cluster [6] [7]
- High-Level-Synthese erzielt ungefähr gleiche Ergebnisse wie ein VHDL Entwurf
- „unroll“, „pipeline“ und „array_partition“ Direktiven haben größten Einfluss [8]

[6] Choi u. a., 2014: “Map-reduce processing of k-means algorithm with FPGA-accelerated computer cluster”

[7] Lavenier, 2000: “FPGA implementation of the k-means clustering algorithm for hyperspectral images”

[8] Winterstein u. a., 2013: “High-level synthesis of dynamic data structures: A case study using Vivado HLS”

Anwendungsfall

- Bildsegmentierung im RGB Farbraum
  - 6 Cluster
  - dreidimensionaler Vektor
  - intuitive Kontrolle durch Visualisierung
  - 814 x 560 Pixel
- Reconfigurable Cloud Computing Framework (RC2F) [9] als Zielumgebung

[9] Knodel u. a., 2015: "Computing framework for dynamic integration of reconfigurable resources in a cloud"

verfügbare Ressourcen (VIRTEX-7 485T)

Tabelle: verfügbare Ressourcen der RC2F Regionen

	total	Single	Dual	Triple	Quad	Quint	Hexa
BRAM 18K	2.060	100	200	300	400	500	600
DSP48E	2.800	340	660	980	1.320	1.660	1.940
LUT	303.600	30.800	60.400	90.000	120.800	151.600	188.400
FF	607.200	61.600	120.800	180.000	241.600	303.200	376.800

10

[10] Knodel u. a., 2017: “Virtualizing Reconfigurable Hardware to Provide Scalability in Cloud Architectures”

k-Means Funktion

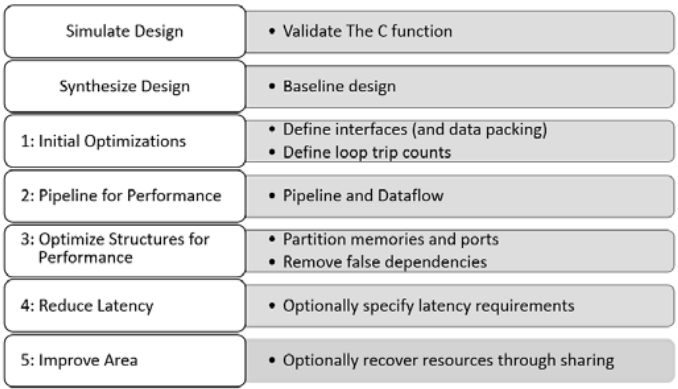
- Parameter:
  - Vektorgröße
  - Bildgröße (Spalten/Zeilen)
  - Clusteranzahl
- Datentypen:
  - RGB Werte: 8Bit (unsigend char)
  - Berechnungen: 32Bit (long)

k-Means Funktion

```
void kMeans(data, result){
    loadData(data, result);
    do{
        copyOldCentroids();
        for(alle Pixel){
            for(alle Centroids){
                calcDistance(Pixel, Centroid);
            }
            accumulateMeans(assignment, pixel);
        }
        calcNewCentroids()
    }while(newCentroids!=oldCentroids);
}
```

3 Entwurfsprozess (HLS)

Vorgehen



11

[11], 2017: UltraFast High-Level Productivity Design Methodology Guide

Basisimplementierung

Tabelle: Ressourcenverbrauch

Name	BRAM_18K	DSP48	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	642
FIFO	-	-	-	-
Instance	-	-	752	864
Memory	1.026	-	160	12
Multiplexer	-	-	-	334
Register	-	-	495	-
Total	1.026	1	1407	1852
Available	2.060	2.800	607.200	303.600
Utilization (%)	49	∼0	∼0	∼0

Basisimplementierung

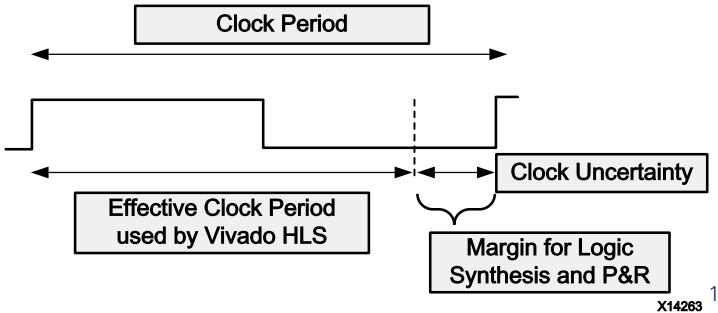
Tabelle: Latenzen in Takten

Latency		Interval	
min	*max*	*min*	*max*
30.085.724	30.086.912	30.085.725	30.086.913

Tabelle: Timing Basisimplementierung

	Target	Estimated	Uncertainty
Clock in ns	10,00	8,55	1,25

Konzept der Unsicherheit



[12]

[12], 2017: Vivado Design Suite User Guide, High Level Synthesis

Schleifenübersicht

Tabelle: Übersicht der Schleifen

Loop Name	Latency		Iteration Latency	Trip Count	Pipelined
	min	*max*			
- load11	3.646.768	3.646.768	8	455.846	no
+ load_12	6	6	2	3	no
+ load_13	6	6	2	3	no
- dowhileBody	26.438.954	26.440.142	26.438.955 ~ 26.440.143	0 ~ 1	no
+ copyCEN11	48	48	8	6	no
++ copyCEN12	6	6	2	3	no
+ calcPixel	26.438.720	26.438.720	58	455.840	no
++ calcPCEN	48	48	8	6	no
+++ calcD	6	6	2	3	no
++ saveCenSum	6	6	2	3	no
+ dowhileBody.3	48	48	8	6	no
++ cenOut	6	6	2	3	no
+ compare11	48	48	8	6	no
++ compare12	6	6	2	3	no

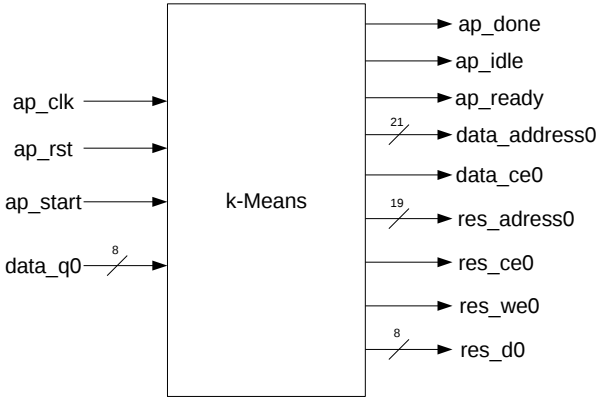
- = 1. Ebene + = 2. Ebene ++ = 3. Ebene

Schleifenübersicht

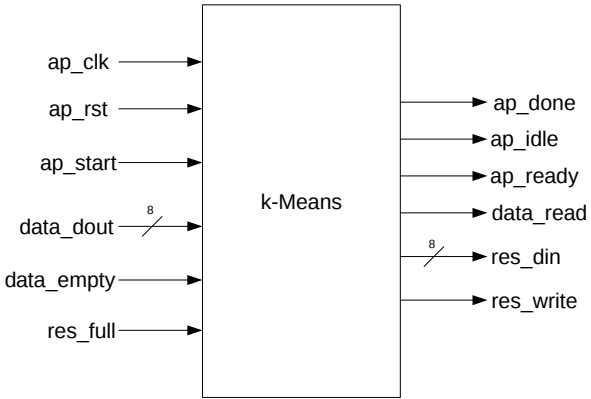
Tabelle: Übersicht der Schleifen

Loop Name	Latency		Iteration Latency	Trip Count	Pipelined
	min	*max*			
- load11	3.646.768	3.646.768	8	455.846	no
+ load_12	6	6	2	3	no
+ load_13	6	6	2	3	no
- dowhileBody	26.438.954	26.440.142	26.438.955 ~ 26.440.143	0 ~ 1	no

Basisinterface



Interface angepasst



Direktive: „PIPELINE“

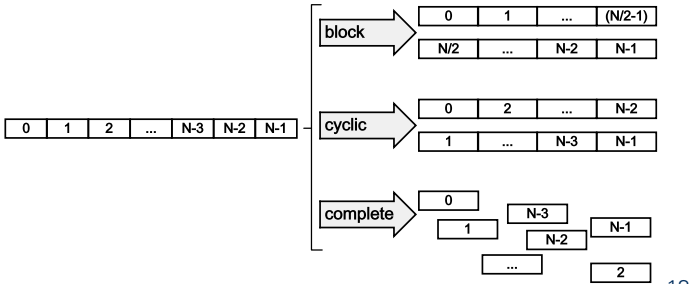
- für Schleifen und Funktionen
- Verwendung: von Innen nach Außen
- nutzt automatisch die „unroll“ Direktive
- Optionen:
  - *Iteration Interval* (*II*)
  - *enable loop rewinding*
  - *enable flushing*
  - *disable loop pipelining*
- globales Flag möglich

Direktive: „PIPELINE“

Tabelle: Vergleich des Ressourcenverbrauchs

	FIFO	*Pipeline*	*Inline 10ns*	*Inline 20ns*
BRAM_18K	1.026	773	774	774
DSP48	1	3	18	18
FF	1.407	1.577	9.323	9.177
LUT	1.852	2.007	9.475	9.473

Direktive: „ARRAY_PARTITION“



13

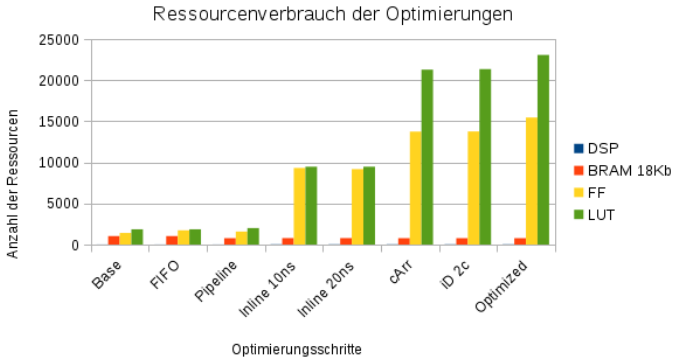
[13], 2017: Vivado Design Suite User Guide, High Level Synthesis

Direktive(GUI): „config_core“

- IP Cores werden ohne Pipeline verwendet
- bei bestimmten IP Cores können mit Pipelines initialisiert werden
- z.B. der DSP48 Kern (mit einer Latenz von 4 Takten)
- wirkt sich sehr positiv auf das Scheduling der Synthese aus

4 Auswertung

Optimierungsschritte:



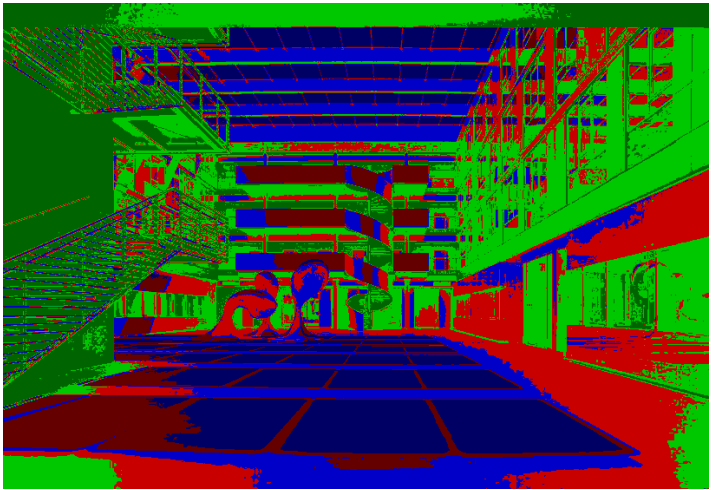
Simulation

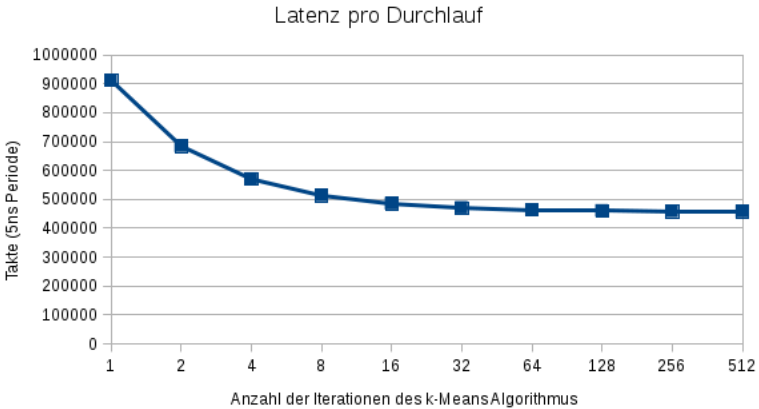
- RTL/Co Simulation
- erzeugt automatisch eine VHDL Testbench aus der C++ Testbench
- Datenausgaben werden separat gespeichert
- Waveform Ausgabe für Ein- und Ausgänge [14]

[14], 2017: Vivado Design Suite User Guide, High Level Synthesis

Abbildung: original Testbild

Abbildung: Ergebnis der Simulation




Latenz pro Durchlauf
Takte (5ns Periode)
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
1
2
4
8
16
32
64
128
256
512
Anzahl der Iterationen des k-Means Algorithmus

4 Auswertung

Schleifenübersicht

Tabelle: Übersicht der Schleifen

	Latency			Initiation Interval			
Loop Name	*min*	*max*	*Iteration Latency*	*achieved*	*target*	*Trip Count*	*Pipelined*
- load11	455.847	455.847	2	1	1	455.846	yes
- dowhileBody	455.990	4.559.909	455.991	-	-	1 ~ 10	no
+ copyCEN11	6	6	1	1	1	6	yes
+ calcPixel	455.851	455.851	13	1	1	455.840	yes
+ newCEN11_newCEN12	87	87	71	1	1	18	yes
+ L_cenOut	18	18	2	1	1	18	yes
+ compare11_compare12	18	18	2	1	1	18	yes

4 Auswertung

Clusteranzahl

Tabelle: Anpassung der Clusteranzahl

		K=2	*K=3*	*K=4*	*K=5*	*K=6*	*K=7*	*K=8*	*K=16*	*K=32*	*K=64*
BRAM_18K		768	768	768	768	768	768	768	768	768	2.060
DSP48		6	9	12	15	18	21	24	48	96	1
FF		10.915	11.106	12.052	14.277	15.461	16.399	17.054	23.240	37.851	80.739
LUT		11.887	14.174	16.156	19.454	23.082	25.990	29.520	38.497	85.393	401.834
Clock	*Target*	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00	5,00
	Estimated	4,87	5,39	5,39	5,57	4,36	4,26	4,44	4,26	4,26	4,33
Latenz	*min*	911.793	911.805	911.816	911.829	911.840	911.853	911.864	911.956	912.140	183.248.529
	max	5.015.325	5.015.436	5.015.537	5.015.649	5.015.759	5.015.871	5.015.972	5.016.820	5.018.516	1.824.279.000

Clusteranzahl

Tabelle: Anpassung der Clusteranzahl (Ausschnitt)

		K=2	*K=3*	*K=4*	*K=5*
BRAM_18K		768	768	768	768
DSP48		6	9	12	15
FF		10.915	11.106	12.052	14.277
LUT		11.887	14.174	16.156	19.454
Clock	*Target*	5,00	5,00	5,00	5,00
	Estimated	4,87	5,39	5,39	5,57
Latenz	*min*	911.793	911.805	911.816	911.829
	max	5.015.325	5.015.436	5.015.537	5.015.649

4 Auswertung

Ergebnis

- Bildbeispiel: 80 Iterationen
- Taktperiode: 5 ns (200 MHz)
- Gesamtlatenz: 36.935.130
- = 184,6 ms

- C++ Sourcecode des k-Means Algorithmus
- i5-4690 (3,5GHz)
- 1 Kern
- = 3,704 s

→ FPGA Implementierung 20x schneller als eigene C++ Implementierung

Vivado HLS

Vorteile:

- effizientes Pipelining
- schnelles Anpassen der Hardware
- bessere Lesbarkeit

Nachteile:

- keine vollständige High-Level-Synthese (Direktiven, Optionen)
- Fachkenntnisse für Hardware und Synthese notwendig
- Details manchmal schwer umsetzbar
- keine Unterstützung für Sourcecode Änderungen

Ausblick

- Testen einer modularen Variante
- weitere Untersuchungen zum Verhalten
- Regeln für Synthese aufstellen bzgl. Anwendungen und Ressourcen
- Entwicklung einer Software für die automatisierte Synthese des k-Means Algorithmus
- Neuronale Netze für automatische High-Level-Synthese