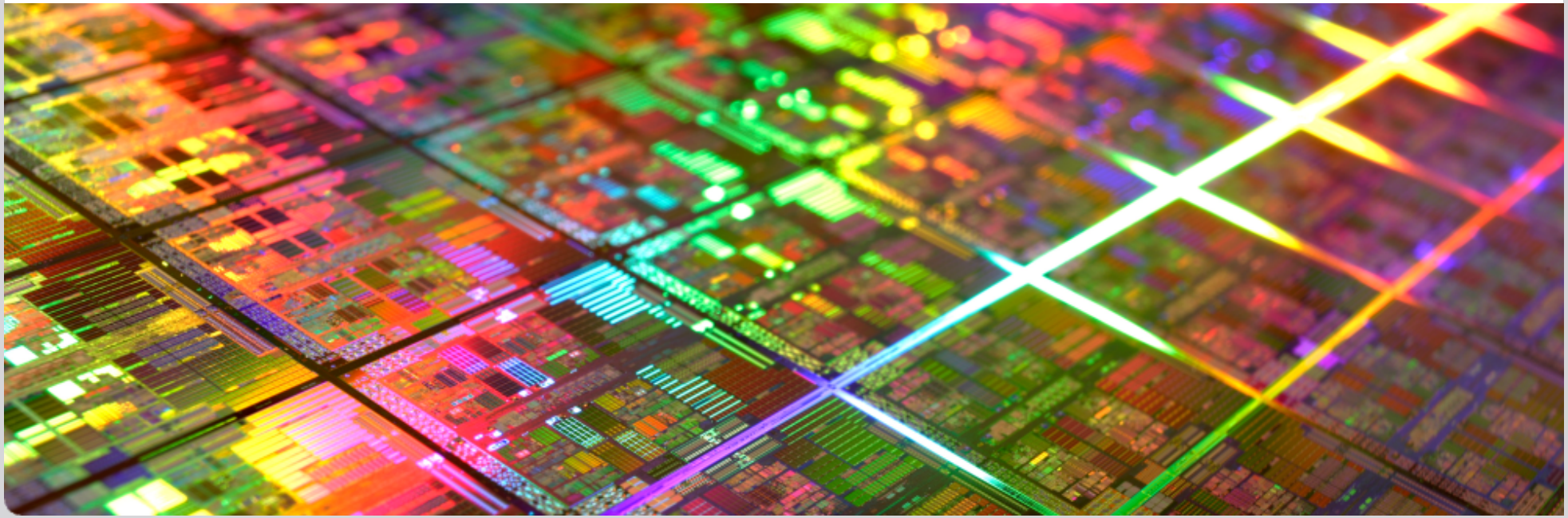
Karlsruher Institut für Technologie

Zentralübung Rechnerstrukturen im SS 2013
Vektorrechner

Mario Kicherer, Prof. Dr. Wolfgang Karl

Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

18. Juli 2013



Übersicht

Karlsruher Institut für Technologie

Letzte Übung

- Vektorprozessoren
- Übungsaufgaben zu Vektorprozessoren

Vektorrechner

Karlsruher Institut für Technologie

Definition

- Unter einem Vektorprozessor (Vektorrechner) versteht man einen Rechner mit pipelineartig aufgebautem/n Rechenwerk/en zur Verarbeitung von Arrays aus Gleitpunktzahlen.

Merkmale

- Vektor = Array aus Gleitpunktzahlen
- Vektoreinheit = Satz von Vektorpipelines in einem Rechenwerk
- Vektor- und Skalareinheit können parallel laufen
- Bei Vektoroperationen entfällt die sonst nötige Adressberechnung
- Vektoroperationen können verkettet werden

Vektorrechner

Parallelverarbeitung in einem Vektorrechner

- Vektor-Pipeline-Parallelität
- Mehrere Vektor-Pipelines in einer Vektoreinheit
  - Verkettung
- Vervielfachung der Pipelines
- Mehrere Vektoreinheiten

Vektorrechner

Eigenschaften

- Vektor Stride
  - Stride-Capability notwendig, wenn Daten nicht konsekutiv im Speicher liegen (z.B. bei Standard Matrix-Matrix-Multiplikation).
- Vektor-Maskierungssteuerung/Vektor-Mask-Register
  - Jede Vektorinstruktion arbeitet nur auf den Vektorelementen, deren Einträge eine 1 haben.
  - Z.B. zur Auflösung von IF-Anweisungen

Vektorrechner

Karlsruher Institut für Technologie

Aufgabe 1.1

Implementieren Sie mit den gegebenen Vektorbefehlen die DAXPY-Operation $Y = a \cdot X + Y$.
Welches Programmierkonstrukt wird durch diese Vektoroperation ersetzt?

Vektorrechner

Vektorbefehle I

Instruktion	Operanden	Funktion
ADDV.D	V1,V2,V3	Add elements of V2 and V3, then put each result in V1.
ADDVS.D	V1,V2,F0	Add F0 to each element of V2 , then put each result in V1.
SUBV.D	V1,V2,V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS.D	V1,V2,F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV.D	V1,F0,V2	Subtract elements of V2 from F0, then put each result in V1.
MULV.D	V1,V2,V3	Multiply elements of V2 and V3, then put each result in V1.
MULVS.D	V1,V2,F0	Multiply each element of V2 by F0, then put each result in V1.
DIVV.D	V1,V2,V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS.D	V1,V2,F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV.D	V1,F0,V2	Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.
LVWS	V1,(R1,R2)	Load V1 from address at R1 with stride in R2, i.e., R1+i x R2
SVWS	(R1,R2),V1	Store V1 from address at R1 with stride in R2, i.e., R1+i x R2
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1+V2(i), i.e., V2 is an index.

Vektorrechner

Vektorbefehle II

Instruktion	Operanden	Funktion
SVI	(R1+V2),V1	Store V1 to vector whose elements are at R1+V2(i), i.e., V2 is an index.
CVI	V1,R1	Create an index vector by storing the values 0, 1 x R1, 2 x R1, ..., 63 x R1 into V1
S--V.D S--VS.D	V1,V2 V1,F0	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S--VS.D performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MTC1 MFC1	VLR,R1 R1,VLR	Move contents of R1 to the vector-length register Move the contents of the vector-length register to R1
MVTM MVFM	VM,F0 F0,VM	Move contents of F0 to the vector-mask register Move contents of vector-mask register to F0.

Vektorrechner

DAXPY: $Y = a \cdot X + Y$ berechnet in Double-Precision.

Aufgabe 1.1

```
LV       V1, Rx      ; load vector X
MULVS.D  V2, V1, F0  ; vector-scalar multiply
LV       V3, Ry      ; load vector Y
ADDV.D   V4, V2, V3  ; vector add
SV       Ry, V4      ; store result vector
```

Ersetztes Konstrukt

Die Schleife, die nötig ist um über die Vektoren zu iterieren entfällt im Fall der Vektorrechnung.

Aufgabe vergl. Hennessy Patterson, Computer Architecture a Quantitative Approach, 2nd Edition, B-8 – B-10

Vektorrechner

Karlsruher Institut für Technologie

Aufgabe 1.2

Gruppieren Sie voneinander unabhängige Vektorbefehle der DAXPY-Berechnung in sogenannte **Convoys** und stellen Sie eine Ausführungsreihenfolge dieser Gruppen auf. Gehen Sie davon aus, dass jede Vektorfunktionseinheit nur einmal existiert. Die Vektorinstruktionen der jeweiligen Gruppe werden parallel zur Ausführung gebracht und benötigen zur Ausführung jeweils ein sogenanntes **chime**. Wie viele **chimes** pro FLOP werden insgesamt benötigt?

Aufgabe vergl. Hennessy Patterson, Computer Architecture a Quantitative Approach, 2nd Edition, B-10

Vektorrechner

Aufgabe 1.2 (Lösung)

Convoys

1. `LV V1, Rx`
2. `MULVS.D V2, V1, F0,`
   `LV V3, Ry`
3. `ADDV.D V4, V2, V3`
4. `SV Ry, V4`

Folglich werden 4 Convoys benötigt. Da jeder dieser Convoys nach der Aufgabenstellung ein **chime** zur Ausführung benötigt, braucht man auch 4 **chimes**.
Zusammen mit 2 FLOP pro Ergebnis ergibt sich damit eine Rate von $\frac{\textbf{chimes}}{FLOP}$ von 2.

Vektorrechner

Aufgabe 1.3

Die Ausführungszeit einer Folge von Vektoroperationen hängt auch von der Zeit für das Aufsetzen der Operationen ab. Dieser Overhead ist in der nebenstehenden Tabelle gegeben. Geben Sie für jeden der Convoys aus Aufgabe 1.2 und einer Vektorlänge *n* die folgenden Zeitpunkte an:

- den Startzeitpunkt,
- den Zeitpunkt an dem das erste Ergebnis des jeweiligen Convoys geliefert wird,
- den Zeitpunkt des letzten Ergebnisses.

Einheit	Start-up Overhead
Load/Store Unit	12 cycles
Multiply Unit	7 cycles
Add Unit	6 cycles

Wie verhält sich für n=64 diese Betrachtung zur Abschätzung mittels **chimes** aus Aufgabe 1.2?

Vektorrechner

Aufgabe 1.3 Lösung

Convoy	Startzeit	Erstes Ergebnis	Letztes Ergebnis
1. LV	0	12	$11 + n$
2. MULVS, LV	$12 + n$	$12 + n + 12$	$23 + 2n$
3. ADDV	$24 + 2n$	$24 + 2n + 6$	$29 + 3n$
4. SV	$30 + 3n$	$30 + 3n + 12$	$41 + 4n$

Aufgabe vergl. Hennessy Patterson, Computer Architecture a Quantitative Approach, 2nd Edition, B-11

Vektorrechner

Aufgabe 1.3 Lösung (fortges.)

Die Zeit pro Ergebnis ist für einen Vektor der Länge 64: $\frac{41+4*64}{64} = 4 + (41/64) = 4,64$ Zyklen. Verglichen mit der Schätzung von 4 **chimes** ergibt sich, dass die genauere Rechnung aufgrund des Overheads für das Aufsetzen der jeweiligen Operationen um $\frac{4,64}{4} = 1,16$ mal höher ausfällt.

Aufgabe vergl. Hennessy Patterson, Computer Architecture a Quantitative Approach, 2nd Edition, B-12 mod errata 3

Vektorrechner

Karlsruher Institut für Technologie

Aufgabe 1.4

Vektorbefehle werden oft durch ein spezielles Speichersystem mit Verschränkung (memory interleaving) und mehreren Speicherbänken unterstützt. Wie lange dauert ein Ladebefehl eines 64-elementigen Vektors bei 16 Speicherbänken und einer Latenz von 12 Zyklen

- mit einem Stride von 1,
- bei einem Stride von 32?

Aufgabe vergl. Hennessy Patterson, Computer Architecture a Quantitative Approach, 2nd Edition, B-21

Vektorrechner

Aufgabe 1.4 Lösung

- Stride von 1:
  Latenz von 12 Zyklen für erstes Element und 63 Zyklen für alle weiteren zu holenden Elemente, da bei 16 Speicherbänken jeder nächste Zugriff auf die jeweils nächste Bank geht und somit die Latenz von 12 Takten versteckt werden kann. Folglich benötigt man 75 Zyklen oder 1,17 Takte pro Element.
- Stride von 32:
  Da 32 ein Vielfaches von 16 (der Anzahl der Bänke) ist, handelt es sich hier um den schlechtesten Fall. Jeder Zugriff des Strides geht auf die gleiche Memory Bank und kollidiert mit dem vorhergehenden. Damit benötigt jeder Speicherzugriff die Latenz von 12 Takten und man benötigt insgesamt:
  $12 * 64 = 768$ Takte oder 12 Takte pro Element.

Vektorrechner

Karlsruher Institut für Technologie

Aufgabe 1.5

Um die Abarbeitung der Vektorbefehle zu beschleunigen, haben Sie in der Vorlesung die Verkettung (engl. chaining) von Vektoroperationen kennengelernt. Vergleichen Sie die Ausführung mit und ohne Verkettung der folgenden Instruktionssequenz miteinander:

```
MULTV V1, V2, V3
ADDV  V4, V1, V5
```

Die Vektoren haben 64 Elemente und die Verzögerung des Additionseinheit - und der Multiplikationseinheit sind 6 und 7 Zyklen. Wie groß ist der erzielte Speedup?

Aufgabe vergl. Hennessy Patterson, Computer Architecture a Quantitative Approach, 2nd Edition, B-25/24

Vektorrechner

Aufgabe 1.5 Lösung

```
MULTV V1, V2, V3
ADDV  V4, V1, V5
```

	Verzögerung
Multiplikationseinheit	7 Zyklen
Additionseinheit	6 Zyklen

Ohne Verkettung

```
 7  |  63  |  6  |  63  |
MULTV       ADDV
```

Gesamt 139 Takte

Mit Verkettung

```
 7  |  63  |
    |  6   |  63  |
```

Gesamt 76 Takte

Vektorrechner

Karlsruher Institut für Technologie

Aufgabe 1.5 Lösung (fortgesetzt)

$$Speedup_{Verkettung} = \frac{139 \text{ Takte}}{76 \text{ Takte}} \approx 1,83$$

Somit wird durch die Verkettung der zwei Operationen bei einer Vektorlänge von 64 Elementen bereits ein Speedup von $1,83$ erzielt.

Vektorrechner

Aufgabe 1.6

```
int i;
int a[n], b[n];
for (i = 0; i < n; i++)
{
    if (a[i] < b[i])
    {
        b[i] = i;
    }
}
```

1. Realisieren Sie dieses Code-Fragment mittels Vektorbefehlen. Gehen Sie bei Ihren Überlegungen davon aus, dass ein Vektorregister je alle *n* Werte der Arrays *a* oder *b* aufnehmen kann.

Aufgabe 1.6 (forts.)

Lösung - Initialisierung

```
MTC1 VLR, R1          # vector-length register := n
                      # wobei R1 den Wert n enthaelt
LV   V1, Ra           # int a[n] in V1 laden
LV   V2, Rb           # int b[n] in V2 laden
mov  R1, 1            # R1 mit 1 initialisieren
CVI  V3, R1           # Create Vektor Index
                      # 0, R1 * 1, R1 * 2, ...
                      # entspricht for (i=0;i<n,i++)
```

Aufgabe 1.6 (forts.)

Lösung - Berechnung

```
SLTV.D  V1, V2   # compare elements with 'Less Than'
                 # if true 1 in Vektor Mask Register
                 # else 0 in Vektor Mask Register
                 # if (a[i] < b[i])

SUBV.D  V2, V2, V2   # Komponenten von V2 mit 1
                     # im VMR werden auf 0 gesetzt

ADDV.D  V2, V2, V3   # diese Komponenten werden mit
                     # den Werten aus V3 aufgefuellt
                     # B[i] = i;
```

Aufgabe 1.6 (forts.)

Lösung - Abschluss

```
CVM             # Clear Vektor Mask
                # alle Eintraege im VMR auf 1 setzen

SV   Rb, V2     # alle Komponent von V2
                # an Adresse in Rb speichern
                # entspricht Schreiben von b[i]
```

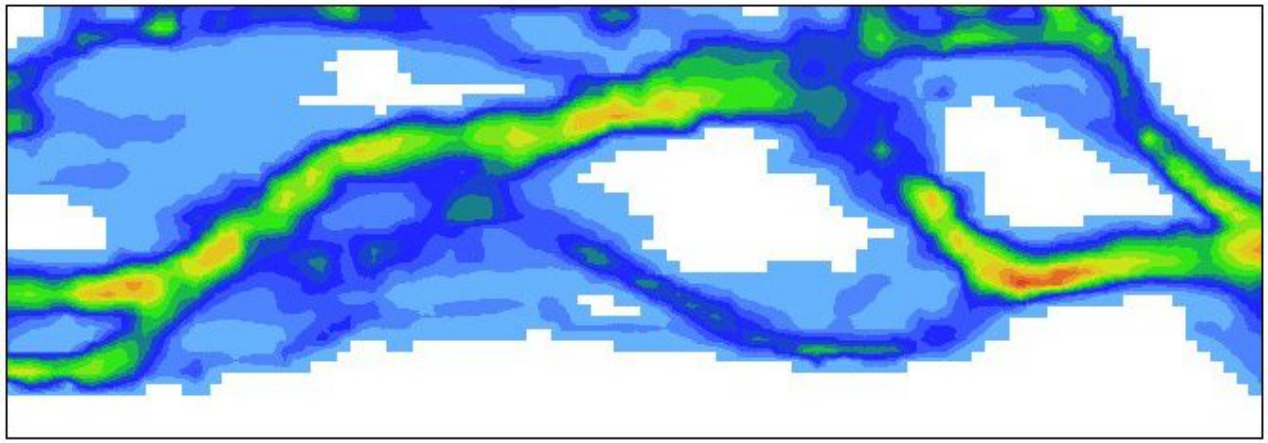
Arbeiten für Studenten I

Bisher:

- Serielle Ausführung seit >10 Jahren

Herausforderung:

- Anpassung an heterogene Parallelsysteme
- Adaptives, unstrukturiertes Gitter
- Modellabhängige Genauigkeitsanforderungen
- Einhaltung von Zeitschranken trotz hoher Dynamik



Arbeiten für Studenten II

3D-Ultraschall-Computertomographie (USCT) am Institut für Prozessdatenverarbeitung und Elektronik (IPE), Campus Nord

Ziel

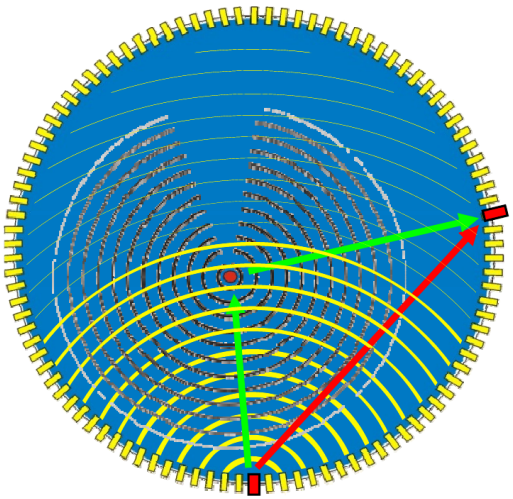
Schnelle schmerz- und strahlungsfreie Untersuchung

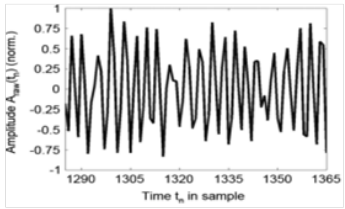
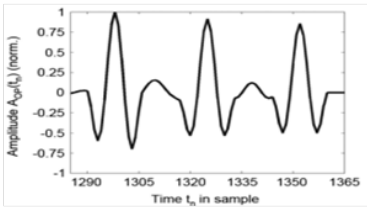
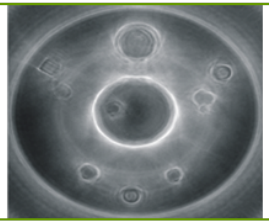
Arbeiten für Studenten II

Karlsruher Institut für Technologie

Funktionsprinzip:

- Halbkugel mit Sensoren
- Jeder Sensor kann senden und empfangen
- Zusammenführung und Analyse des empfangenen Schalls



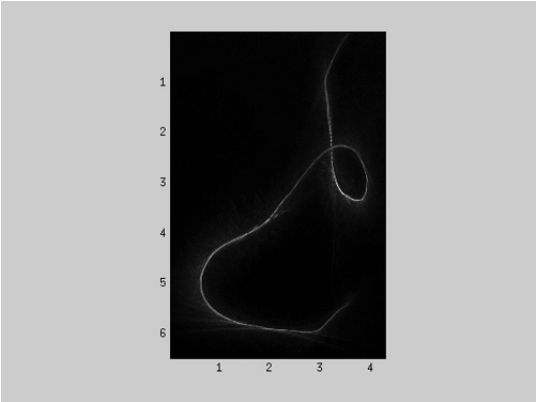
1. Signalaufzeichnung	2. Signalverarbeitung	3. Bildrekonstruktion
		
FPGA (80x Cyclone II)	PC (12 CPU-Kerne) MATLAB	4x GPUs (GTX 590) CUDA C
3,5 Mio. A-Scans = 17 min	½ A-Scans 2957,8 A-Scans/s = 10 min	½ A-Scans * 1024² x 64 106,5 GVoxels/s = 18 min

Arbeiten für Studenten II

Herausforderung:

- Verschiedene Vorgehensweisen möglich zur Vorverarbeitung und Rekonstruktion
- plus: unterschiedliche Implementierungen und Datenformate

→ Welche Kombination am schnellsten auf heterogenem Parallelsystem?



Draht mit 0,7 μm Durchmesser

Karlsruher Institut für Technologie

Zentralübung Rechnerstrukturen im SS 2013
Vektorrechner

Mario Kicherer, Prof. Dr. Wolfgang Karl

Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

18. Juli 2013