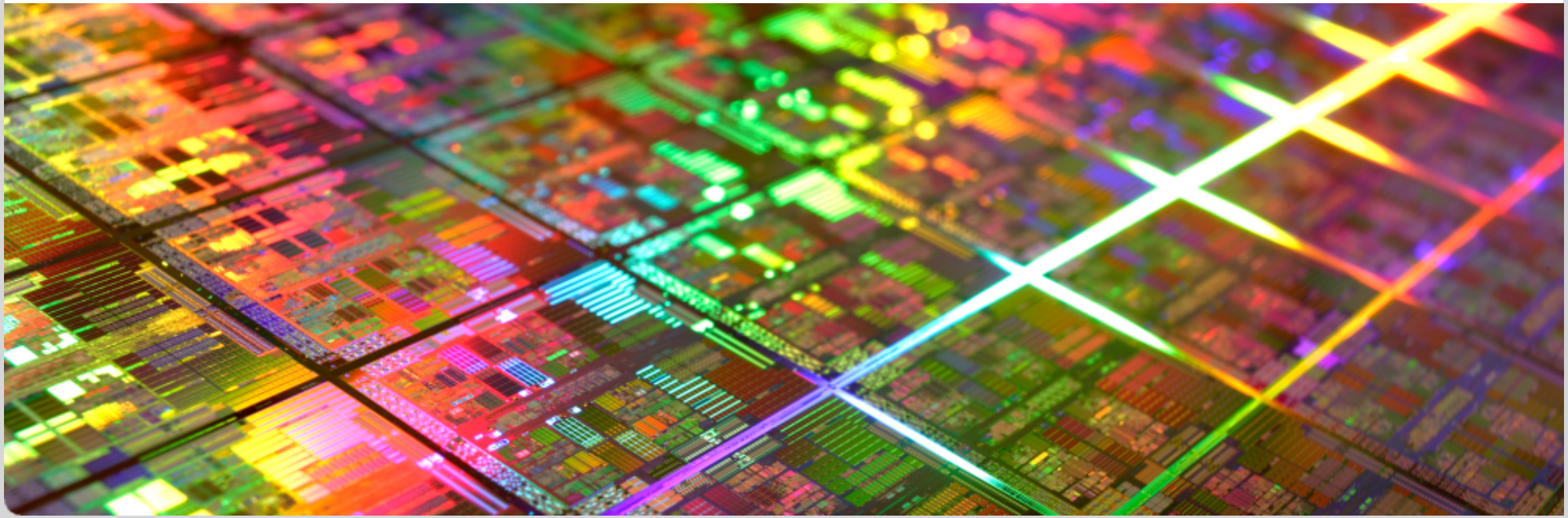
Karlsruher Institut für Technologie

Zentralübung Rechnerstrukturen im SS 2014
Vektorrechner

Oliver Mattes, Prof. Dr. Wolfgang Karl

Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

17. Juli 2014



Überblick

Karlsruher Institut für Technologie

Inhalt der Übung:

- Vektorprozessoren
- Aufgabe 1

- Überblick über den Stoff aller Übungen
- Hinweise zur Klausur

- Forschung am Lehrstuhl CAPP

- **Evaluation der Zentralübung Rechnerstrukturen**

Vektorrechner

Definition

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

Merkmale

- SIMD-Prinzip
- Vektor = Array aus Gleitpunktzahlen
- Eine oder mehrere Vektorpipelines
- Ein Satz von Vektorpipelines bildet die Vektoreinheit
- Bei Vektoroperationen entfällt Adressberechnung
- Zusätzlich eine Skalareinheiten (mehrere Funktionseinheiten)
- Vektor- und Skalareinheit können parallel laufen

Vektorrechner

Parallelverarbeitung in einem Vektorrechner

- Vektor-Pipeline-Parallelität
  - Überlappung durch mehrere Stufen in einer Pipeline
- Mehrere Vektor-Pipelines in einer Vektoreinheit
  - mehrere, meist funktional verschiedene Vektorpipelines

  $\Rightarrow$ Verkettung
- Vervielfachung der Pipelines
  - Vervielfachung gleicher Vektorpipelines
  - Ausführung eines Befehls gleichzeitig auf mehreren Pipelines
  - vielfach schneller, aber verfielfachter Hardware-Aufwand
- Mehrere Vektoreinheiten
  - Vektor-Multiprozessor oder Parallel-Vektorrechner
  - MIMD-/SIMD-Prinzip

Vektorrechner

Karlsruher Institut für Technologie

Parallele Speicherzugriffe

(n-fache) Speicherverschränkung / Memory Interleaving

- Unterteilung des Speichers in n Speicherbänke
- Verteilung von Speicherplatz A_i auf Speicherbank M_j wenn $j = i \mod n$
- nach einer Anlaufzeit werden in jedem Speicherzyklus n Speicherworte geliefert

Vektor Stride

- Vektor Stride wird verwendet, wenn Daten nicht fortlaufend im Speicher liegen (z.B. bei Standard Matrix-Matrix-Multiplikation).
- Vektor Stride gibt Abstand zwischen Elementen an, die in einem Register verwendet werden

Vektorrechner

Verkettung

- Verkettung mehrere verschiedener Vektorpipelines zur Ausführung einer Folge von Vektoroperationen.
- die Ergebnisse einer Pipeline werden sofort der nächsten Pipeline zur Verfügung gestellt

Vektor-Maskierungssteuerung / Vektor-Mask-Register

- Jede Vektorinstruktion arbeitet nur auf den Vektorelementen, deren Einträge eine 1 haben.
- z.B. zur Auflösung/Umgehung von nicht vektorisierbaren IF-Anweisungen

Vektorrechner – Aufgabe 1

a) **Implementieren Sie mit den Vektorbefehlen aus dem Foliensatz der Vorlesung die DAXPY-Operation:**

$$Y = a \cdot X + Y$$

wobei X und Y Vektoren sind und a ein Skalar.

Welches Programmierkonstrukt wird durch diese Vektoroperationen ersetzt?

Vektorrechner – Aufgabe 1

Karlsruher Institut für Technologie

Vektorbefehle:

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 – Aufgabe 1

Karlsruher Institut für Technologie

Vektorbefehle:

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 – Aufgabe 1

a) **DAXPY:** $Y = a \cdot X + Y$ (berechnet in Double-Precision)

```
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
```

Welches Programmierkonstrukt wird durch diese Vektoroperationen ersetzt?

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 – Aufgabe 1

b) **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

Karlsruher Institut für Technologie

b) 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 $\frac{chimes}{FLOP}$ von 2.

Vektorrechner – Aufgabe 1

c) **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 1b) 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 Zyklen
Multiply Unit	7 Zyklen
Add Unit	6 Zyklen

Wie verhält sich für n=64 diese Betrachtung zur Abschätzung mittels chimes aus Aufgabe 1b)?

Vektorrechner – Aufgabe 1

c) Zeitpunkte der Convoys:

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

c) **Wie verhält sich für n=64 diese Betrachtung zur Abschätzung mittels chimes aus Aufgabe 1b)?**

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 – Aufgabe 1

d) **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

Karlsruher Institut für Technologie

d) Ladebefehl eines 64-elementigen Vektors, 16 Speicherbänke, Latenz von 12 Zyklen

- **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 Speicherbank 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 – Aufgabe 1

Karlsruher Institut für Technologie

d) Ladebefehl eines 64-elementigen Vektors, 16 Speicherbänke, Latenz von 12 Zyklen

- **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 Speicherbank 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 – Aufgabe 1

e) **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

e)

```
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 – Aufgabe 1

Karlsruher Institut für Technologie

e) **Wie groß ist der erzielte Speedup?**

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

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

Vektorrechner – Aufgabe 1

Karlsruher Institut für Technologie

f) **Gegeben Sei folgendes Fragment eines C-Programmes:**

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

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.

Vektorrechner – Aufgabe 1

Karlsruher Institut für Technologie

f) 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++)
```

Vektorrechner – Aufgabe 1

Karlsruher Institut für Technologie

f) 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;
```

Vektorrechner – Aufgabe 1

Karlsruher Institut für Technologie

f) 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]
```

Überblick über den Stoff aller Übungen

Karlsruher Institut für Technologie

Übung 1

- Chip-Fertigung
- Fertigungskosten
- Schaltungsentwurf mit VHDL

Übung 2

- Low-Power-Entwurf, Leistungsverbrauch
- Leistungsbewertung
  (Architektur, Rechenanlage)
- Benchmarks

Überblick über den Stoff aller Übungen

Karlsruher Institut für Technologie

Übung 3

- Fehlertoleranz
- Redundanzsysteme
- Sprungvorhersage
  (Sättigungs-/Hysteresezähler, n-Bit Prädiktoren)

Übung 4

- Pipelining
- Superskalartechniken
- Algorithmus von Tomasulo
- VLIW-Prozessoren

Überblick über den Stoff aller Übungen

Karlsruher Institut für Technologie

Übung 5

- Parallelismus
- Parallelrechner und Quantitative Maßzahlen
- Parallele Programmierung
  (Parallelisierungsprozess, Programmiermodelle)

Übung 6

- Verbindungsstrukturen
  (Charakterisierung, Statische/dynamische Verbindungsnetze)
- Vergleich von Parallelrechnern

Überblick über den Stoff aller Übungen

Karlsruher Institut für Technologie

Übung 7

- Cacheorganisation und -eigenschaften
- Cachekoheränzprotokolle (MESI, MOESI)
- DSM-Systeme

Übung 8

- Vektorprozessoren

Hinweise zur Klausur

Organisatorisches

- Klausur: 6. August 2014, 11:00 Uhr
- Hörsäle: Daimer und Benz
- **Sitzverteilung:**
  wird kurz vor Klausur auf Homepage veröffentlicht!

- 60 Punkte, 60 min $\Rightarrow$ 1 Punkt pro Minute
- Kurze Antworten, keine Romane

- Aufgaben werden zusammen durchgelesen
- Bearbeitung beginnt erst danach

Hinweise zur Klausur

Karlsruher Institut für Technologie

Tips

- Aufgaben mit Abfrage von Begriffen oder Formeln
- Aufgaben mit viel Text und Berechnung
- Aufgaben mit Anwendung von Algorithmen, Methoden
- Schaubilder, Grafiken,. . .

- Schwerpunkt der Teilgebiete der Aufgaben variabel

- Alte Klausuren durchrechnen! (siehe Homepage)
- Stil von Aufgaben bleibt ähnlich
- Variabilität im Inhalt der Vorlesung

Hinweise zur Klausur

Karlsruher Institut für Technologie

Lernmaterial

- Vorlesungfolien
- Übungsmaterialien (Folien, Übungsblatt, Lösungen)
- Angegebene Literatur aus Vorlesung und Übung
- Alte Klausuren (siehe Homepage)

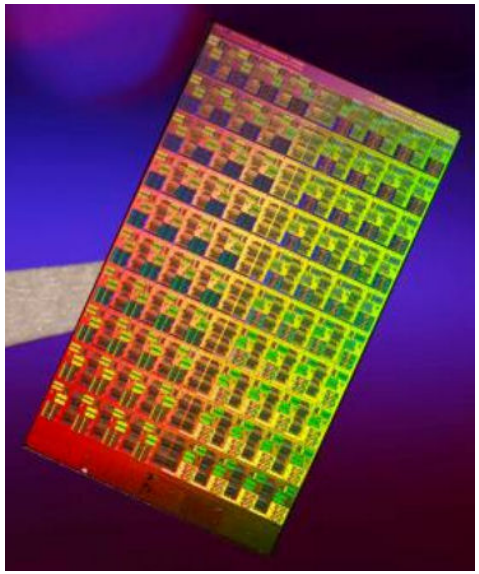
Karlsruher Institut für Technologie

Fragen?

Abschlussarbeiten – Self-aware Memory

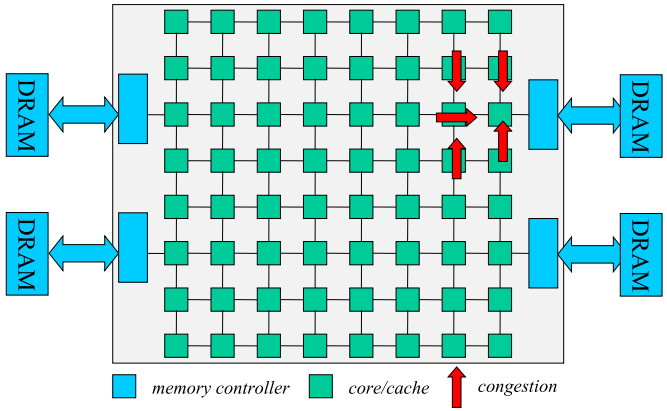
Motivation:

- Trend zu Manycore-Systemen, Heterogenen Architekturen
- Neue hochdynamische Anwendungsszenarien
- Skalierbare Speicherarchitektur notwendig
- Dynamische Adaption an aktuell ausgeführte Anwendungen



Stand der Technik:

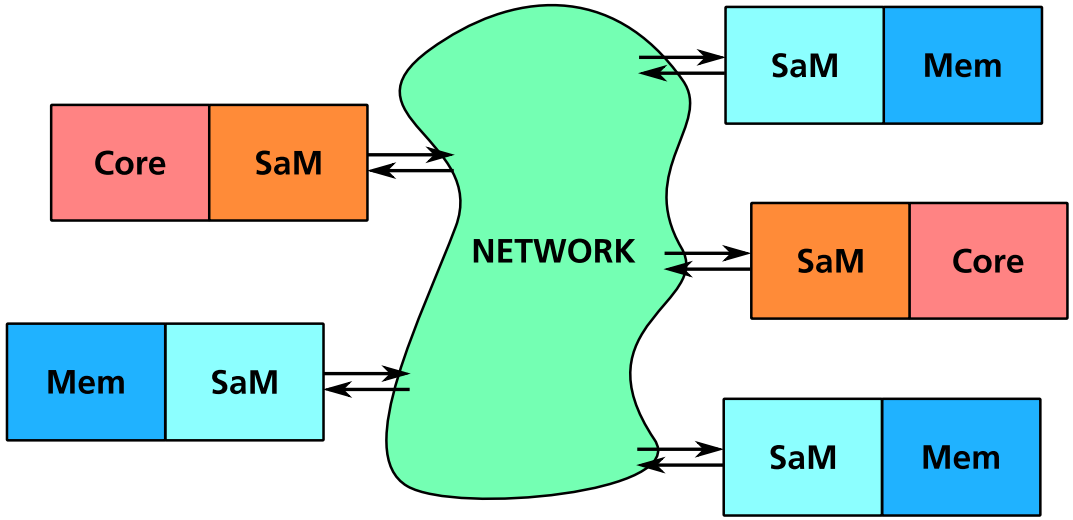
- Zentrale, sehr statische Verwaltung
- Ineffiziente Speicherzuteilung
- Gegenseitiges Blockieren/Behinderung beim Zugriff auf externen Speicher



Abschlussarbeiten – Self-aware Memory

Dezentrale, adaptive Speicherarchitektur:

- Dynamische Speicherverwaltung und -zuordnung
- Dezentraler Aufbau, autonome Komponenten
- Skalierbarkeit, Fehlertoleranz



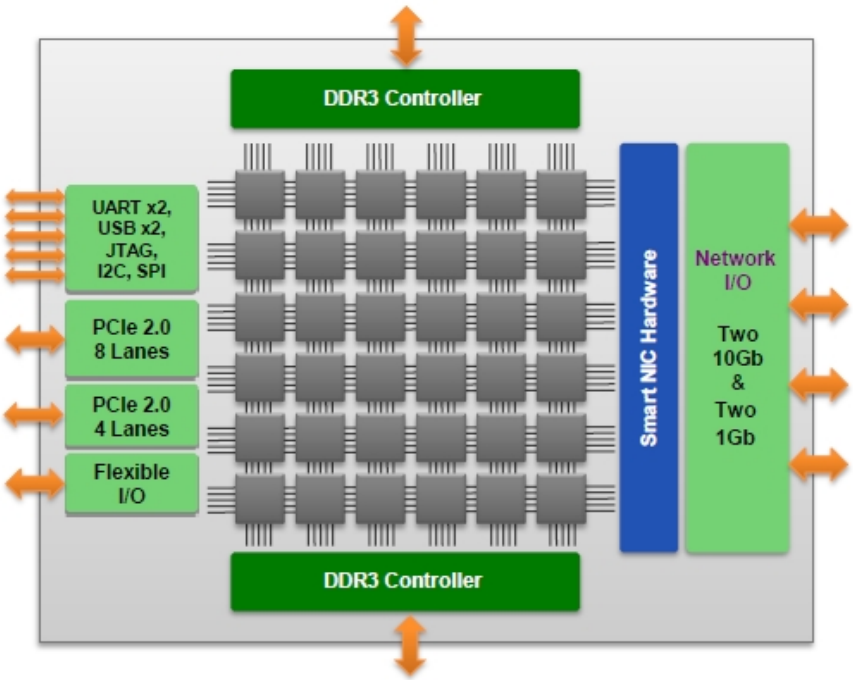
- Private und Shared-Memory-Unterstützung
- Speicherkonsistenz, -Kohärenz, Transactional Memory
- Fortlaufende, dezentrale Selbst-Optimierung

Abschlussarbeiten – Self-aware Memory

Karlsruher Institut für Technologie

Evaluation:

- **SystemC-basierte Software-Simulation**
- Hardware-Implementierung mit FPGA-Karten
- Software-Implementierung als Serverdienst
- **Tilera Tile-GX 36-Kern-System**
- **NUMAScale System** (CC-NUMA mit globalem Adressraum)



Abschlussarbeiten – Self-aware Memory

Karlsruher Institut für Technologie

Mögliche Themen:

- Hoch-dynamische Anwendungsszenarien und ihre Auswirkung auf den Selbst-Optimierungs-Zyklus
- Untersuchung der Skalierbarkeit von SaM und Vergleich mit existierenden Manycore-Architekturen
- **Portierung von SaM auf Tilera Tile-GX-Architektur (36-Kern Prozessor)**
- **Portierung von SaM auf numascale-System (ccNUMA-Architektur)**
- Cache-Nutzung mit SaM unter Wahrung der Cache-Kohärenz
- Untersuchung verschiedener Netzwerke/-architekturen: Photonic Networks, 3D-stacked Memory,...
- Flexible Speicherzuordnung in Embedded Systemens

Abschlussarbeiten – Self-aware Memory

- **Bei Interesse an Abschlussarbeit (BC/MA) zu Self-aware Memory bei Oliver Mattes melden!**
  - E-Mail, Telefon, persönlich,. . .

- Abschlussarbeiten auch in weiteren Forschungsgebieten am Lehrstuhl CAPP möglich:
  - Parallele Programmierung und Architekturen
  - Transactional Memory
  - Nutzung und Programmierung von heterogene Systemen
  - FPGA-basierte Hardware-Beschleunigung

Evaluation

Evaluation der Zentralübung Rechnerstrukturen

- **Evaluation der Vorlesung erfolgte getrennt!**
- **Evaluation der Übung!**
  - Übungsleitung
  - Übungsblätter, Übungsfolien
  - Organisation
- Freitextantworten mit konstruktiver Kritik!

$\Rightarrow$ Verbesserung der Übungen

- **Ausgefüllte Evaluationsbögen bitte vorne abgeben!**

Karlsruher Institut für Technologie

Zentralübung Rechnerstrukturen im SS 2014
Vektorrechner

Oliver Mattes, Prof. Dr. Wolfgang Karl

Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

17. Juli 2014