Vektorrechner und SIMD

Stephan Rabanser

27. Mai 2014

Inhaltsverzeichnis

Vektorrechner — Überblick

- Alternativbegriffe: Vektor-, Array- oder Feldprozessor
- CPUs können Daten in Vektorform abarbeiten
- Gegenstück zu Skalarrechnern
- Pseudo-parallele Verarbeitung
- Spezielle Befehle und Register

Funktionsweise

- Pipelining des Befehlszykluses

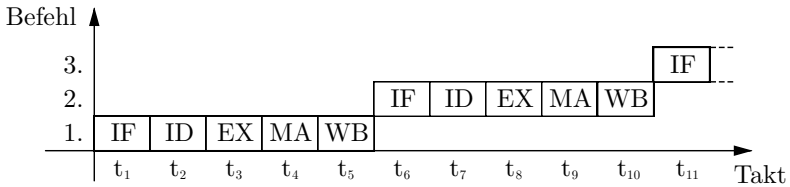


Abbildung 1 : Abarbeitung ohne Pipelining

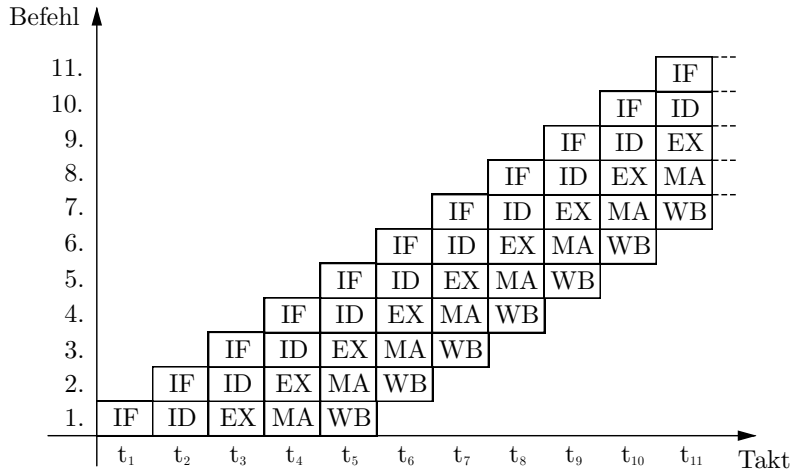


Abbildung 2 : Abarbeitung mit Pipelining

Funktionsweise — Forts.

- Pipelining der Daten

Algorithmus 1 : Skalar

for $i = 0$; $i < n$; $i{+}{+}$ **do**
    Lese nächsten Befehl;
    Dekodiere Befehl;
    Hole erste Zahl;
    Hole zweite Zahl;
    Addiere beide Zahlen;
    Schreibe das Ergebnis zurück in den Speicher;
end

Algorithmus 2 : Vektor

Lese nächsten Befehl;
Dekodiere Befehl;
Hole diese n Zahlen im Bereich a-b;
Hole diese n Zahlen im Bereich c-d;
Addiere diese Zahlen;
Schreibe die Ergebnisse zurück in den Speicher;

Geschichtliche Entwicklung

- 1960: Projekt Solomon von Westinghouse Electric
  - Vielzahl an Co-Prozessoren
  - Steuerung durch skalare CPU
- 1962: ILLIAC IV von der University of Illinois
  - Geplant: 1 GFLOPS, 256 ALUs
  - Schlussendlich: 100 - 150 MFLOPS, 64 ALUs
  - Trotz Schwierigkeiten schnellster Vektorrechner
  - Viele getrennte ALUs $\rightarrow$ Massively Parallel Computing
- 1966: Advanced Scientific Computer (ASC) von Texas Instruments
  - Erstmals Pipelining eingesetzt, auch mehrere Pipelines möglich
  - 20 - 80 MFLOPS
  - Verarbeitung von Vektor- und Skalarbefehlen

Geschichtliche Entwicklung — Forts.

- 1966: STAR-100 von Control Data Corporation
  - Deutlich schwächer als der ASC
  - Lange Initialiserungs- und Ladezeiten
  - Preiswert und platzsparend
- 1976: Cray-1 von Cray
  - Acht spezielle Vektorregister
  - Anwendung der Befehle zwischen diesen Registern $\rightarrow$ sehr schnell
  - Pipelining getrennt für unterschiedliche Befehlsklassen $\rightarrow$ vector chaning
  - 80 - 240 MFLOPS je nach Auslastung der Pipelines

Geschichtliche Entwicklung — Forts.

- 1980: ETA-10 von Control Data Corporation
  - Performance im Vergleich zum Cray-1 wenig überzeugend
  - Verkaufszahlen enttäuschend
  - Rückzug von CDC aus dem Supercomputer-Markt
- 1980er: Fujitsu, Hitachi und Nippon Electric Corporation (NEC)
  - Ebenfalls geringe Verbesserungen zum Cray-1
  - Leistung ähnlich, aber deutlich kleiner
- 1980er: Cray-2, Cray X-MP und Cray Y-MP von Cray
  - Cray festigt seine Vormachtstellung

Geschichtliche Entwicklung — Forts.

- 1990er: Vermehrt massively parallel processing (MPP)
  - Sehr viele ALUs auf einem Rechner
  - Exotische Speicherarchitekturen
- 2000er: IBM Virtual Vector Architecture (ViVA)
  - Zusammenfassen von Skalarprozessoren zu einem Vektorprozessor

Aktueller Stand

- Pure Vektorrechner aus den Top500 verschwunden
- Inzwischen 85 % Clusterrechner (MIMD)
  - Gutes Preis-Leistungs-Verhältnis
  - Flexibel in der Nutzbarkeit
- 15 % MPP-Rechner (MIMD)
- Rückkehr von reinen Vektorrechnern unwahrscheinlich
- Erweiterungen als Co-Prozessoren für MIMD-Rechner
  - Meist SIMD-Einheiten

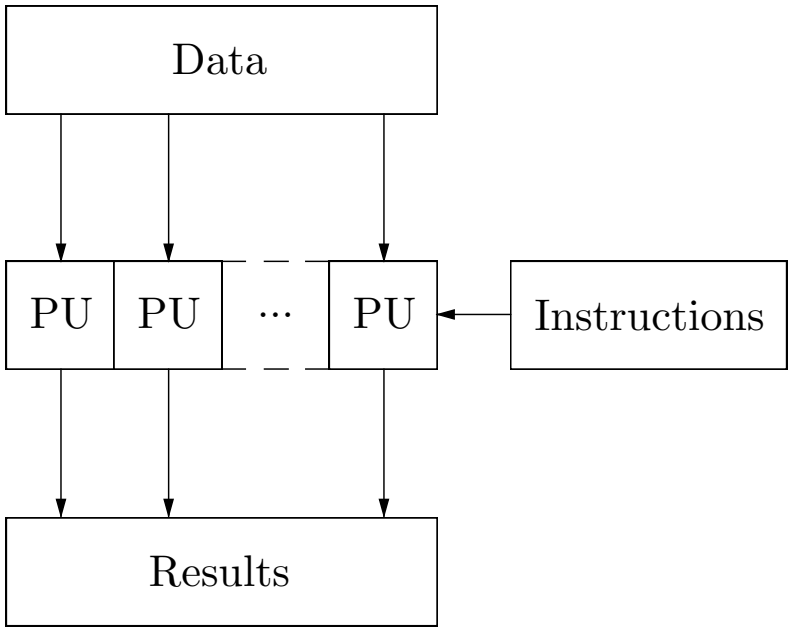
SIMD — Überblick

- SIMD: Single Instruction Multiple Data
- Einteilung nach Flynn'scher Klassifikation

		Data	
		Single	Multiple
Instruction	Single	SISD	**SIMD**
	Multiple	MISD	MIMD

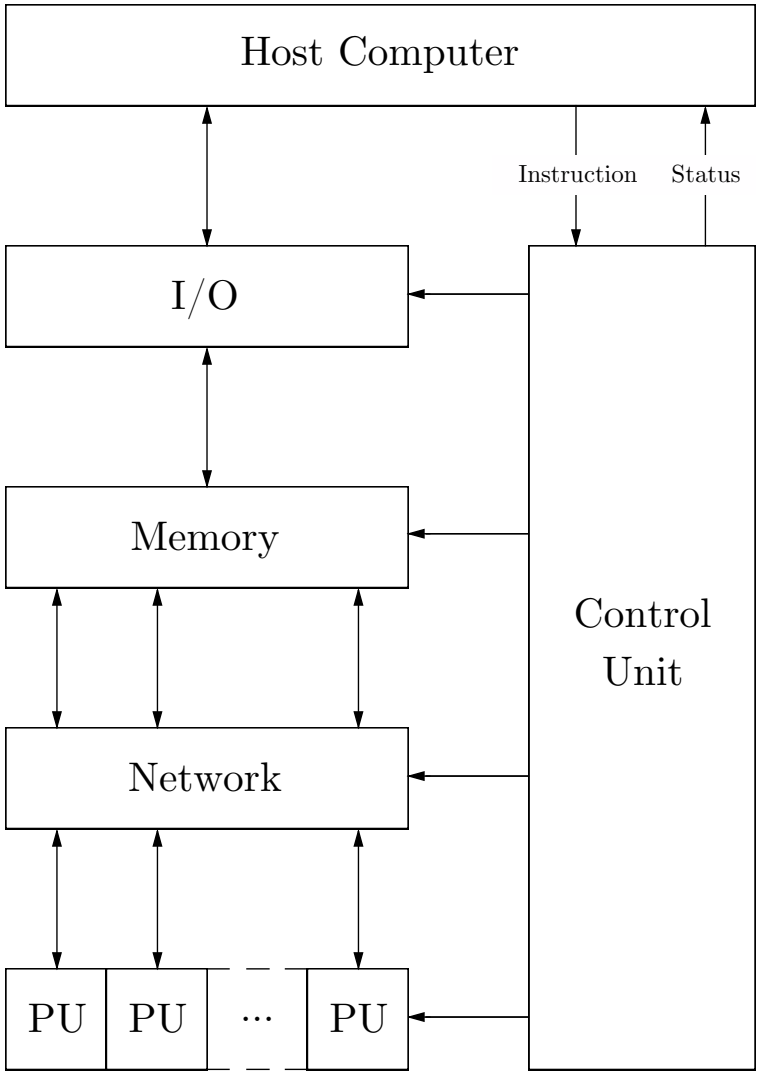
Grundlegende Funktionsweise

- Mehrere Datenströme, ein Befehlsstrom
- Parallelismus auf Datenebene



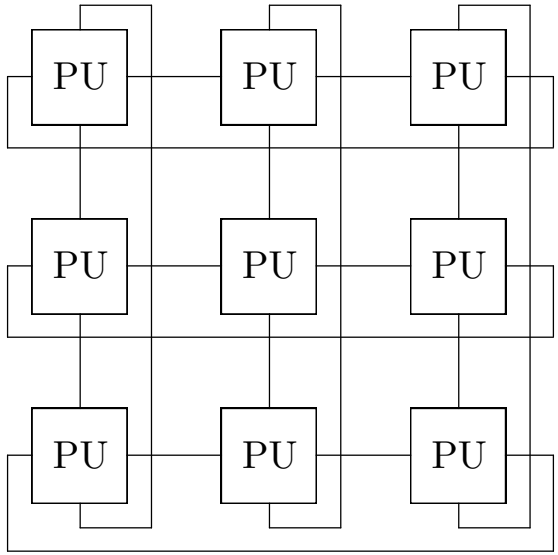
Detailiertere Funktionsweise

- Kontrolleinheit ließt und dekodiert Befehle
- Weiterleitung an Processing Units (PUs)
- Speicher mit genau so vielen Datenpfaden wie Prozessoren
- Spezielles Datennetz für Routing in beide Richtungen
- Über I/O Komponente an Host-Computer angebunden
- Kommunikation zwischen Kontrolleinheit und Host-Computer durch Status- und Befehlskanal
- Heute oft als Co-Prozessor an die CPU angebunden oder direkt auf dem Die


Host Computer
Instruction
Status
I/O
Memory
Network
PU
PU
···
PU
Control
Unit

Detailiertere Funktionsweise — Forts.

- Verschiedene Topologien für Prozessorarray möglich
- Meist Gitterstruktur
  - Positiv: Hohe örtliche Lokalität
  - Negativ: Relativ hoher Durchmesser
    - Durch umschließende Kanten verbesserbar



Gegenüberstellung: SIMD - Vektor

SIMD	Vektor
Parallelität ist in Soft- und Hardware gegeben (echt parallel dank mehrerer ALUs).	Parallelität ist in der Software, aber nicht in der Hardware vorhanden (pseudo-parallel).
Ein Befehl bleibt immer nur für einen CPU-Zyklus erhalten.	Ein Befehl bleibt generell für mehrere CPU-Zyklen erhalten.
Befehlsausführungen passieren zeitgleich auf allen Datensätzen.	Die Befehlsausführung geschieht über Pipelining.
Die Datenelemente werden möglicherweise einzeln geladen, somit sind mehrere Ladevorgänge nötig.	Mehrere Datenelemente können meist durch einen einzelnen Befehl geladen werden.
Eher schmal implementierte Befehle und Register.	Meist breit implementierte Befehle und Register.
Vektoren sollten im Speicher richtig ausgerichtet werden, da dadurch weniger Ladeoperationen von Nöten sind.	Vektoren müssen nicht zwingend im Speicher ausgerichtet werden, die einzelnen Elemente aber schon.

Geschichtliche Entwicklung

- Anwendung in Vektorrechnern der frühen 1970er
- Vektor- getrennt von SIMD-Rechnern betrachtet
- Modernere SIMD-Rechner: “massively parallel” Rechner
- Begeisterung rund um 1990 stark gedämpft
- Einsatz in breiter Masse: Multimedia-Extensions (MMX), 3DNow!, AltiVec und Streaming SIMD Extensions (SSE)
- Die Rolle Apples

Aktueller Stand

- Abkehr von puren SIMD-Rechnern bei Supercomputern
  - SIMD-Erweiterungen auf dem Die oder als Co-Prozessor
- SIMD in aktuellen Prozessoren: AVX
- Von Larrabee zu Many Integrated Core (MIC), Xeon Phi und Nvidia Tesla
- Cell-Prozessor in der PlayStation 3
- General Purpose GPUs (GPGPU) und OpenCL
  - CPU-Code auf die CPU auslagern
- Single Instruction Multiple Threads (SIMT)
  - Erweiterung von SIMD mittels Multithreading

Aktueller Stand — Forts.

- SIMD within a register (SWAR)
- SIMD für das Web mittels Dart
- Künftige Integration in JavaScript

Vor- und Nachteile

- Vorteile
  - Sehr schnelle Abarbeitung von Vektor-Algorithmen
  - Iteration durch Datensätze (Bsp.: for-Schleife)
  - Anwendungsbreiche: Bild, Video, Audio, HPC
- Nachteile
  - Nicht alle Algorithmen eignen sich für eine Vektorverarbeitung
  - Verzweigungen (Bsp.: if-else-Konstrukte)
  - Compiler optimieren noch nicht immer einwandfrei
  - Ausrichtung der Vektoren im Speicher entscheidend für Performance

Fragen?