Vektorrechner und SIMD

Proseminar High Performance Computing (HPC)

Stephan Rabanser
Fakultät für Informatik
Technische Universität München
E-Mail: rabanser@cs.tum.edu

27. Mai 2014

***Zusammenfassung*—Diese Ausarbeitung befasst sich mit den Datenverarbeitungsmethoden "Vektor" und "Single Instruction Multiple Data" (SIMD). Es wird auf deren Anwendung in Computern, deren geschichtliche Entwicklung, sowie deren Funktionsweise eingegangen.**

***Schlüsselworte*—Vektor, Pipelining, Solomon, ILLIAC IV, STAR-100, ASC, Cray, ETA-10, MPP, ViVA, SIMD, MMX, 3DNow!, AltiVec, SSE, AVX, MIC, GPGPU, OpenCL, SIMT, HPC.**

I. Einleitung

Vektor-Verarbeitungsmethoden spielen bereits seit einigen Jahrzehnten bei Computern eine wichtige Rolle. Für wissenschaftlich wichtige Anwendungsfälle, aber auch im Multimedia-Sektor (Bild, Video, Ton) ist Vektorverarbeitung bereits seit langem nicht mehr wegzudenken. Besagte Methoden lassen sich im Wesentlichen in "Vektor" und "SIMD" unterteilen. Im Folgenden werden diese beiden Verarbeitungsmethoden in ihrer geschichtlichen sowie aktuellen Entwicklung beschrieben und auf ihre Funktionsweise hin untersucht.

II. Vektorrechner

A. Überblick

Als Vektorrechner, auch Vektor-, Array- oder Feldprozessor genannt, bezeichnet man CPUs, welche Befehle auf einem eindimensionalen Feld (Vektor) ausführen können. Derartige Rechner können daher Befehle auf mehrere Datensätze anwenden. Vektorrechner stehen im Gegensatz zu Skalarrechnern. Letztere sind lediglich in der Lage einen Befehl auf einen einzelnen Datensatz anzuwenden.

B. Funktionsweise

Ein wichtiges Merkmal von Vektorrechner ist die sehr ausgedehnte Nutzung des Pipelinings. So ziemlich jeder moderne Prozessor, egal ob skalar- oder vektor-orientiert, baut auf Pipelining um den Befehlszyklus möglichst schnell abzuarbeiten. Der grundlegende Ablauf eines solchen Befehlszykluses lautet wie folgt: 1) IF - Instruction Fetch 2) ID - Instruction Decode 3) EX - Execute 4) MA - Memory Access 5) WB - Write Back. Sämtliche Maschinenbefehle lassen sich in diese fünf Phasen aufspalten.[1]

Ohne Pipelining würden die Befehle wie in Abbildung 1 dargestellt von der CPU abgearbeitet.

Man kann leicht erkennen, dass die CPU bei diesem Ansatz sehr schlecht ausgelastet ist. Außerdem sieht man, dass die tatsächliche Ausführung des Befehls lediglich $^1/_5$ der Zeit beansprucht, die restlichen $^4/_5$ werden für Speicherzugriffe und Interpretation aufgewandt.

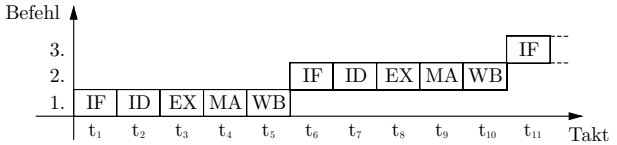

Abbildung 1: Abarbeitung des Befehlszykluses **ohne** Pipelining

Rechner, die Pipelining einsetzen, verfügen daher über spezielle, der CPU unterstehenden, Einheiten, welche nun eben jene restlichen Teilaufgaben des Befehlszykluses erledigen, wodurch die CPU nur mehr für das tatsächliche Ausführen zuständig ist. Wird ein Befehl geladen, so kommt er zuerst in die IF-Einheit und wird nach erfolgreichem Laden an die ID-Einheit weitergeleitet. Nun ist die IF-Einheit wieder frei und kann bereits jetzt mit dem Holen des nächsten Befehles beginnen. Setzt man dies analog für sämtliche folgenden Einheiten fort, so lässt sich der Befehlszyklus (idealerweise) nach dem Fließband-Prinzip, und daher viel effizienter, abarbeiten. Alle Einheiten werden somit konstant voll ausgenutzt.

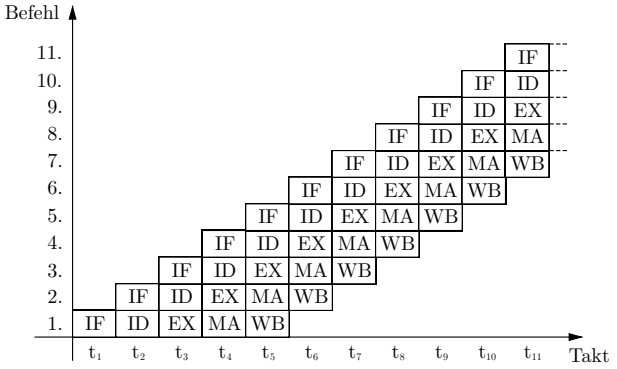

Abbildung 2: Abarbeitung des Befehlszykluses **mit** Pipelining

Eine derartige Abarbeitung birgt allerdings auch Gefahren - sogenannte Pipeline-Hazards - in sich. So tritt bspw. ein Hazard auf, falls ein Befehl das Ergebnis des vorherigen Befehls benötigt.

Vektorrechner tragen das Pipelining nun noch eine Stufe weiter, indem sie nicht nur den Befehlszyklus, sondern auch die Daten, die verarbeitet werden sollen, pipelinen. Die Vektorbefehle sind so implementiert, dass sie ohne viel Aufwand auf einem Datenvektor ihren Dienst verrichten. Ist der Da-

tenbereich erst festgelegt und das erste Datenelement gelesen, so weiß der Befehl, dass das nächste Datenelement an der nächsten Speicherstelle zu finden ist und setzt den Befehl somit automatisch bis zum Ende des angegebenen Bereiches fort. Dies ermöglicht deutliche Zeitersparnisse bei der Dekodierung.

An Hand des folgenden Beispiels lassen sich die Zeitersparnisse gut erkennen:

Algorithmus 1 : Skalar-Algorithmus

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

In jedem Schleifendurchlauf müssen also der Befehl und die Zahlen neu geladen werden, was viel Zeit in Anspruch nehmen kann.

Algorithmus 2 : Vektor-Algorithmus

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;

Dadurch, dass wir in Algorithmus 2 keine Schleife benötigen, ersparen wir uns viele Dekodierungen und Speicherzugriffe.

Um jedoch derartige Vektorbefehle überhaupt verwenden zu können, müssen CPU und unmittelbar daran anknüpfende Komponenten speziell dafür ausgerichtet werden, was unvermeidlich der Architektur Komplexität hinzufügt. Damit lässt sich auch die Tatsache erklären, dass nicht vektor-optimierter Code auf Vektorrechnern in der Regel nicht schneller, sondern oft sogar langsamer läuft.

Zusammenfassend können wir feststellen, dass Vektorrechner somit pseudo-parallel arbeiten. IF und ID sind für jede Vektoroperation nur einmal nötig, der Befehlszyklus wird somit für alle folgenden Operationen auf $^3/_5$ seiner ursprünglichen Länge reduziert.

C. Geschichtliche Entwicklung

Vektorrechner erblickten das erste Mal in den 1960ern das Licht der Welt und prägten nachhaltig die Entwicklung der Supercomputer bis in die 1990er.

Der erste Rechner, der in der Lage war Vektorelemente zu verarbeiten, ging in den frühen 1960ern aus dem Projekt **Solomon** aus dem Hause *Westinghouse Electric* hervor. Bereits dieser Rechner verfügte über eine Vielzahl an Co-Prozessoren, welche jedoch alle unter der Kontrolle einer übergeordneten, klassisch-skalaren CPU standen. Diese CPU leitete dann die Befehle an die ALUs des Rechners weiter. Jeder untergeordnete Vektorprozessor erhielt somit den selben Befehl, konnte diesen aber auf verschiedene Datensätze anwenden. Ein mathematischer Algorithmus, welcher vektorisierbar war, konnte auf diesem Rechner klarerweise deutlich schneller und effizienter arbeiten.[2]

Nur zwei Jahre darauf wurde das Projekt auf Eis gelegt, wenig später allerdings von der *University of Illinois* unter dem Namen **ILLIAC IV** wieder aufgenommen. In der anfänglichen Planung ging man noch von einer Leistung von 1 GFLOPS dank 256 ALUs aus, bei der Fertigstellung rund zehn Jahre später verfügte der Rechner allerdings nur über 64 ALUs und erreichte lediglich 100 - 150 MFLOPS. Das Konzept war jedoch in weiten Teilen stimmig, sodass der ILLIAC IV allen Schwierigkeiten zum Trotz für datenintensive, vektorisierbare Probleme der schnellste Rechner seiner Zeit war. Der Ansatz der getrennten ALUs für jedes Datenelement, welcher im ILLIAC IV Einzug gefunden hatte, wurde von den meisten nachfolgenden Vektorrechnern fallen gelassen und wurde in eine getrennte Kategorie ausgelagert, die als “massively parallel computing” bekannt ist.[3]

Die erste tatsächlich erfolgreiche Umsetzung eines Vektorrechners gelangen *Control Data Corporation* mit dem **STAR-100** und *Texas Instruments* mit dem **Advanced Scientific Computer** (**ASC**) Ende der 1960er. Letzterer bediente sich erstmals des Pipelinings und war in der Lage sowohl Vektor- als auch Skalarbefehle zu verarbeiten. Die Leistung wurde mit 20 MFLOPS beziffert. In der Standardkonfiguration verfügte der ASC über eine einzige Pipe, mittels einer Two-Pipe- oder Four-Pipe-Architektur ließen sich dann dementsprechende 2-fache bzw. 4-fache Leistungssteigerungen erreichen, glücklicherweise ohne an die Grenzen der Speicherbusbandbreite zu stoßen.[4] Im Vergleich zum ASC wirkte der STAR-100 dagegen deutlich weniger überzeugend. Obwohl er bei datenintensiven Anwendungen mit dem hauseigenen CDC 7600 gut mithalten konnte und zudem deutlich preiswerter und kleiner war, enttäuschte er durch teilweise sehr lange Initialisierungs- und Ladezeiten. Um somit einen deutlichen Geschwindigkeitsvorteil überhaupt erkennen zu können, mussten schon sehr speziell angepasste Daten vorhanden sein.[5]

So richtig ausgeschöpft wurde die Vektor-Technik allerdings erst 1976 durch *Cray* mit dem **Cray-1**. Dieser verfügte im Gegensatz zum STAR-100 oder ASC erstmalig über acht spezielle Vektorregister, die jeweils 64 64-Bit Wörter speichern konnten. Da die Befehle somit zwischen Registern angewandt werden konnten und somit der Weg zum Arbeitsspeicher nicht nötig war, konnte der Cray-1 sehr schnell arbeiten. Wie auch schon seine geistigen Vorgänger baute der Cray-1 auf Pipelining um eine schnelle Abarbeitung garantieren zu können, allerdings mit einer sehr effektiven Änderung: Für verschiedene Befehle bzw. Befehlsklassen wurden getrennte Pipelines eingeführt. So wurden bspw. Additionen und Subtraktionen in einer Pipeline abgearbeitet, während Multiplikationen und Divisionen eine andere Pipeline durchliefen. Dieses Aufspalten - auch “vector chaining” genannt - von einer Menge an Vektorbefehlen in kleinere Untermengen und deren nebenläufige Abarbeitung konnte die Performance nochmals steigern. In Zahlen ausgedrückt: Je nach Aufspaltung erreichte der Cray-1 Leistungswerte zwischen 80 MFLOPS und 240 MFLOPS.[6]

Angestoßen von diesen Entwicklungen versuchten sich dann eine Reihe bekannter Konzerne ebenfalls an Vektorrechnern. *Control Data Corporation* versuchte mit dem **ETA-10** erneut in den High-End-Markt einzusteigen, die Verkaufszahlen waren allerdings so enttäuschend, dass sich CDC komplett aus dem Supercomputer-Markt zurückzog.[7] In den 1980ern betraten viele japanische Firmen den Markt, darunter *Fujitsu*, *Hitachi* und *Nippon Electric Corporation* (*NEC*). Die von diesen Unternehmen produzierten Rechner waren dem Cray-1 sehr ähnlich, wobei die Leistung kaum gesteigert, die Größe aller-

dings stark reduziert werden konnte. Trotz aller Bemühungen der Konkurrenz behauptete *Cray* seine Führungsposition im Bereich der Supercomputer und festigte diese weiter durch neue Rechnerentwicklungen, die schließlich zu den Supercomputern **Cray-2**, **Cray X-MP** und **Cray Y-MP** führten. [8][9]

Von da an wurde die Entwicklung der Supercomputer immer mehr vom **massively parallel processing** (**MPP**) geprägt. Im Gegensatz zu einem Cluster-Rechner liegen sämtliche Prozessoren zusammen mit deren ALUs und Speichern in einem Rechner. Um eine gute Ausnutzung dieser örtlichen Lokalität zu erreichen, implementieren diese Rechner oft sehr ausgefallene Speicherarchitekturen, um den schnellen Austausch von Daten zwischen Prozessoren zu erlauben. Umgesetzt wurden MPP-Rechner meist nach dem SIMD- oder MIMD-Prinzip.

Um die Vorteile der Vektorverarbeitung aber auch in andere Architekturen tragen zu können, entwickelte *IBM* die **Virtual Vector Architecture** (**ViVA**). Diese Technologie machte es möglich eine Vielzahl an Skalarprozessoren zu einem Vektorprozessor zusammenzufassen, um somit auf Standard-Prozessoren Vektoranwendungen erheblich zu beschleunigen.[10]

D. Aktueller Stand

Pure Vektorrechner lassen sich heute nicht mehr unter den Top 500 der schnellsten Supercomputer der Welt finden. Sie wurden hauptsächlich von Clusterrechnern verdrängt, welche inzwischen 85 % dieser Top 500 stellen.[11] Dies ist zum einen dem besseren Preis-Leistungs-Verhältnis der aktuellen MIMD-Prozessoren, zum anderen deren überaus flexibler Nutzbarkeit für unterschiedlichste Problemstellungen geschuldet.

MPP-Rechner sind allerdings immer noch sehr stark in den Top 500 mit den restlichen 15 % vertreten. Diese arbeiten jedoch alle nach dem MIMD-Prinzip.[11]

Eine Rückkehr der puren Vektorrechner in die Top 500 gilt derzeit als sehr unwahrscheinlich.[12] Der momentane Trend liegt in der Entwicklung von leistungsstarken Co-Prozessoren für CPU und GPU, die nur dann angesteuert werden, sobald spezielle Anwendungsfälle vorliegen, die besonders für eine Vektorverarbeitung geeignet wären. Diese Einheiten arbeiten allerdings hauptsächlich nach dem SIMD-Prinzip.

III. SIMD

A. Überblick

Single Instruction Multiple Data - kurz SIMD - beschreibt eine Rechnerarchitektur nach der **Flynn'schen Klassifikation**, die 1966 von *Michael J. Flynn* eingeführt wurde. Besagte Klassifikation dient zur Unterteilung von Computer-Architekturen, basierend auf der Anzahl der bereit gestellten Befehls- und Datenströme.[13]

Folgenede Tabelle veranschaulicht die Flynn'sche Klassifikation.

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

B. Funktionsweise

Wie sich bereits aus dem englischen Begriff ableiten lässt, verfügen Rechner, die der SIMD-Architektur entsprechen, über mehrere Datenströme, aber nur einen Befehlsstrom. Der ein und der selbe Befehl kann also zeitgleich auf mehrere Datensätze angewandt werden. SIMD-Rechner ermöglichen somit Parallelismus auf Datenebene, aber keine echte Nebenläufigkeit wie MIMD-Rechner.[14]

Abbildung 3 zeigt die vereinfacht dargestellte Funktionsweise der SIMD-Architektur.

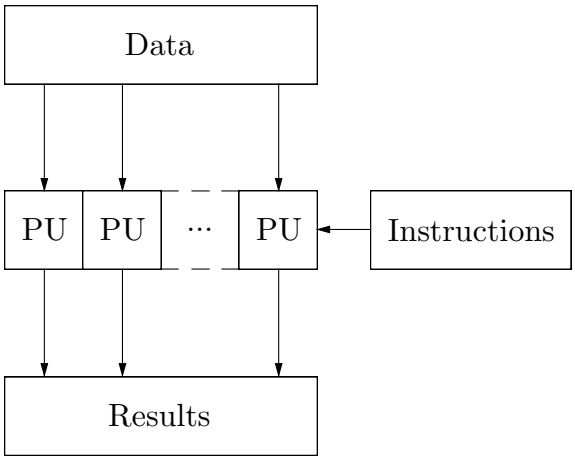

Abbildung 3: Grundlegende Funktionsweise der SIMD-Architektur

Klassischerweise verfügen SIMD-Rechner über eine Kontrolleinheit, welche die Befehle einließt, dekodiert und an die Prozessoreinheiten (PUs - processing units) weiterleitet. Die Daten stammen aus einem Speicher, der über genau so viele Datenpfade, wie Prozessoren verfügt. Zwischen dem Speicher und dem Feld an Prozessoren liegt ein spezielles Datennetz, welches für das Routing in beide Richtungen zuständig ist. Dadurch kann eine erhöhte Flexibilität bei der Wahl des Pfades zwischen Speicher und Prozessorfeld erreicht werden. Generell wird der SIMD-Rechner dann zudem über eine I/O-Komponente an einen Host Computer angebunden. Dieser Host Computer dient unter anderem zur allgemeinen Kontrolle der Vektorprozessoren. Über einen Befehls-Kanal kann der Host Computer Befehle an die Kontrolleinheit des SIMD-Rechners schicken, Statusänderungen der Kontrolleinheit empfängt der Host Computer über einen Status-Kanal.[15]

Viele moderne Rechner verfügen bereits standardmäßig über SIMD-Einheiten. In solchen Fällen sind diese Einheiten entweder als Co-Prozessoren an die CPU angebunden, oder sind direkt auf dem Chip als Untereinheit vorhanden und nahezu unmittelbar ansteuerbar.

Der Zusammenhang zwischen den einzelnen Modulen wird in Abbildung 4 dargestellt.

Die Topologie des Prozessorarrays variiert von Rechner zu Rechner, wobei die meisten SIMD-Rechner die Gitter-Struktur implementieren. Dies liegt vor allem daran, dass dadurch eine hohe örtliche Lokalität erreicht wird. Als negativ gilt allerdings der relativ hohe Durchmesser (maximale direkte Entfernung zwischen zwei Knoten). Dieser negative Aspekt lässt sich jedoch durch das Einfügen von zusätzlichen, umschließenden Kanten verringern. Bei einer Prozessoranzahl von n beträgt der Durchmesser somit $\sqrt{n}$.[16]

Abbildung 5 veranschaulicht die beschriebene Gitterstruktur.

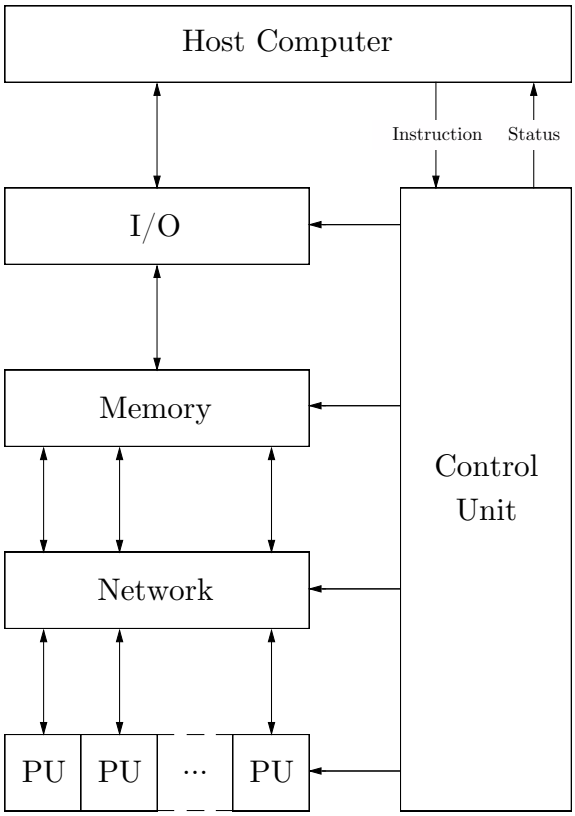

Abbildung 4: Detailiertere Funktionsweise von SIMD

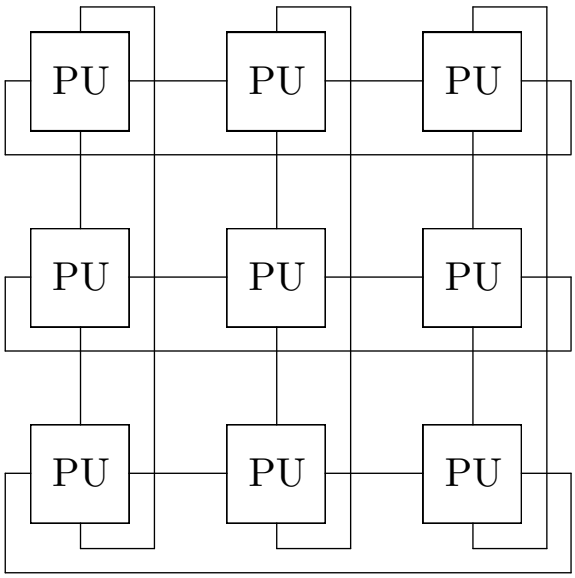

Abbildung 5: Gitterstruktur des Verbindungsnetzes zwischen den PUs

C. Gegenüberstellung: SIMD - Vektor

Nun da wir die Funktionsweisen beider Rechnerarten beleuchtet haben, lassen sich die wichtigsten Unterschiede wie folgt zusammenfassen.

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.

Auch wenn sich SIMD- und Vektor-Technik theoretisch wie oben dargestellt abgrenzen lassen, so gibt es doch auch manchmal Überschneidungen bzw. Mischvarianten, sodass sich hier nicht immer eine harte Linie ziehen lässt und die Begriffe oft auch synonym verwendet werden.

D. Geschichtliche Entwicklung

SIMD-Befehle fanden in den Vektorrechnern der frühen 1970ern ihre erste Anwendung. Bereits diese Rechner waren in der Lage, einen Befehl auf mehrere Datensätze anzuwenden, die in einem Vektor angeordnet waren. Populär wurde der SIMD-Ansatz allerdings erst gegen Ende der 1970er und zu Beginn der 1980er durch Anlagen des US-Supercomputer-Spezialisten *Cray*.

Vektorrechner-Architekturen werden von diesem Zeitpunkt an getrennt von SIMD-Maschinen betrachtet, da Vektorrechner die Vektorelemente über Pipelining auf mehrere Prozessoren verteilen. Moderne SIMD-Rechner verarbeiten jedoch alle Vektorelemente exakt zur selben Zeit. Der auszuführende Befehl bleibt dabei bei beiden Verarbeitungsvarianten immer gleich.[17]

Die erste Ära moderner SIMD-Rechner wurde vorwiegend durch Computer im sogenannten “massively parallel”-Stil geprägt. Dabei handelte es sich um Rechner, welche über mehrere Tausend an parallel arbeitender Prozessoren verfügten, deren Funktionsumfang allerdings im Gegensatz zu heute noch relativ eingeschränkt war. Als einer der bekanntesten Vertreter gilt die **Connection Machine** (**CM**) von *Thinking Machines*, welche bereits in frühen Versionen über deutlich mehr als 60.000 Prozessoren verfügte, die alle den selben Befehl zum selben Zeitpunkt ausführten.

Im Laufe der 1980er und speziell zu Beginn der 1990er schwand jedoch die anfängliche Begeisterung für reine SIMD-Rechner und machte laufend mehr den aufstrebenden MIMD-Prozessoren Platz. SIMD-Prozessoren der aktuellen Generation gingen daher nicht mehr aus dem Supercomputer-, sondern aus dem Desktop-Markt hervor.

In der breite Masse kam SIMD allerdings erst durch die Prozessorhersteller *Intel* und *AMD* zur Anwendung. 1996 stellte Intel erstmals **Multimedia-Extensions** (**MMX**) für die x86 Prozessorarchitektur vor, 1998 folgte AMD mit **3DNow!**. Wenig später zogen dann auch die Konkurrenzhersteller *Apple*, *IBM* und *Motorola* (auch bekannt als *AIM-Allianz*) mit dem **AltiVec**-System nach. Nur drei Jahre später konterte Intel erneut mit den **Streaming SIMD Extensions** (**SSE**), die inzwischen in mehreren Weiterentwicklungen vorliegen und später auch bei AMD 3DNow! weitestgehend ersetzten.[18] Speziell auf x86 Prozessoren bereitete die Umsetzung allerdings zu Beginn Probleme. Teure Kontextwechsel beim Umschalten zwischen FPU- und MMX-Registern, ineffiziente Compiler, zu knapp bemessene Register und unpraktikable Datentypen erzeugten zu Beginn viel Skepsis gegenüber SIMD.

Apple hatte deutlich weniger Probleme bei der Umsetzung, da deren Computer zu jenem Zeitpunkt noch nicht auf die x86-, sondern auf die PowerPC-Architektur setzten und somit AltiVec zur Verfügung stand. Zu diesem Zeitpunkt gab es dafür deutlich bessere Compiler von *Motorola*, *IBM* und *GNU*, die nachträgliche Assembler-Anpassungen durch den Programmierer im Allgemeinen vermieden. Zudem profitierte Apple von der Geschlossenheit ihrer Hard- und Software, sodass sie Programme wie bspw. iTunes und QuickTime speziell für die Apple Hardware optimieren und somit deutlich bessere Leistungen erzielen konnten. Als Apple dann 2006 ihre Computer auf x86-Prozessoren umstellte und sich somit weitgehend der übrigen Industrie anglich, waren die größten Schwierigkeiten auch dort beseitigt. Viele Programme, darunter auch die Programmierumgebung Xcode, wurden dann auf SSE2/SSE3 optimiert.[19] Durch diesen Absprung von Apple verlor IBM zwar einen großen Kunden, AltiVec wurde jedoch laufend weiterentwickelt und ist auch noch heute in den Power-Chips zu finden.

E. Aktueller Stand

Im Supercomputing-Bereich geht der Trend nun schon lange zu MIMD, da derartige Prozessoren bereits seit einiger Zeit immer bessere Leistung für immer weniger Geld bereitstellen.[20] Heutige Prozessoren sind allerdings stets mit starken SIMD-Erweiterungen oder SIMD-Co-Prozessoren ausgestattet, welche allerdings deutlich weniger breit implementiert sind als pure SIMD-Rechner. Oft sind diese Einheiten heute sogar direkt auf dem CPU-Chip vorhanden.

Auch Standard-Desktoprechner sind bereits seit einigen Jahren in der Lage, Computerspiele und -simulationen, sowie Videoverarbeitung zu unterstützen. Daher wuchs der Bedarf für Vektorprozessoren laufend stärker an.

Seit 2011 verbauen *Intel* und *AMD* die **Advanced Vector Extensions** (**AVX**) in ihre Prozessoren. Diese verfügen über Befehle, die 256 Bits gleichzeitig verarbeiten können. Beginnend mit 2015 will Intel dann auf **AVX-512** umsteigen, was - wie der Name bereits erahnen lässt - 512-Bit-Unterstützung mit entsprechenden Befehlen und Registern bringen soll.[21] Intels **Larrabee** Konzept versprach bereits 2010 die Einführung von 512-Bit-Befehlen, auf Grund von andauernden Verspätungen und enttäuschender Performance wurde das Projekt allerdings eingestellt. Wenig später wurde dieses Konzept aber von Intel neu aufgesetzt und führte zur Intel **Many Integrated Core** (**MIC**) Architektur. Die bekannteste Implementierung davon findet sich derzeit im **Xeon Phi**, welcher zusammen mit dem **Tesla** von *Nvidia* als einer der stärksten SIMD-Vektor-Co-Prozessoren auf dem Markt gilt.[22][23] In den Top 10 der schnellsten Supercomputer weltweit sind auf den Plätzen 1, 2, 6 und 7 (Stand: November 2013) Rechner gelistet, welche stark auf eben jene genannten Erweiterungen bauen um ihre volle Rechenleistung zu erreichen.[24]

Speziell mit dem Anbruch des HD-Zeitalters vor fast zehn Jahren wuchsen die Anforderungen an Grafikeinheiten rapide. Diese neue Entwicklung erreichte die Spielekonsolen zuerst, sodass sich *Sony* mit der **PlayStation 3** und *Microsoft* mit der **Xbox 360** gezwungen sahen neue Wege zu gehen. *Sony* entwickelte zusammen mit *Toshiba* und *IBM* den **Cell Prozessor**, welcher über einen 64-Bit-PowerPC-Kern, eine Skalar- sowie sieben SIMD-Einheiten verfügte.[26] Diese Einheiten waren derart leistungsstark, dass damit weltweit mehrere PS3-Cluster, darunter auch an Universitäten und der amerikanischen Luftwaffe, für HPC-Anwendungsfälle in Betrieb genommen wurden.[27] Bereits die Leistung einer einzige PS3-Einheit reichte, um MD5-Hashs mittels der Brute-Force-Methode innerhalb weniger Stunden zu knacken, was für eine inzwischen acht Jahre alte Konsole ein sehr beachtliches Ergebnis darstellt.[28] Damit war eine PS3-Einheit vergleichsweise so stark wie 30 Standard-Rechner zu dieser Zeit.

SIMD-Befehle werden häufig zur Verarbeitung von dreidimensionalen Grafiken verwendet. Da Grafikchips schon lange über integrierte SIMD-Module verfügen, werden diese Aufgaben nun hauptsächlich von Grafikkarten übernommen und entlasten somit erheblich die CPU. Einige Systeme verfügen auch über Funktionen, welche das Zusammenfassen von Daten in Vektoren erleichtern. Speziell Datenkompression und Anwendungsbereiche wie Kryptografie (AES, SHA und RSA) und Multimedia können davon stark profitieren.[25]

Der aktuellste Trend geht nun soweit, dass zunehmend Standard-CPU-Code auf die GPU ausgelagert werden soll, sodass nicht nur Grafik-Anwendungen von der Vektorarchitektur der Grafikkarten profitieren. Die sogenannten **general-purpose-GPU** (**GPGPU**) Technologien wie **OpenCL** sind derzeit auf dem Vormarsch.[29] Dies könnte zu einer deutlich ausgedehnten Nutzung von SIMD-Instruktionen führen. Zusätzlich erweitern viele moderne Grafikeinheiten von *Nvidia* den SIMD-Ansatz durch Multithreading. Derartige GPU-Architekturen werden mit **Single Instruction Multiple Thread** (**SIMT**) bezeichnet.[30]

Doch nicht nur im Supercomputer- oder Standard-Konsumenten-Markt hat sich viel getan. Auch low-end Prozessoren können mittels **SIMD within a register** (**SWAR**) SIMD fähig werden, indem durch eine Reihe von technischen Tricks SIMD-Code in den Allzweckregistern von CPUs ausgeführt werden kann, die eigentlich nicht über eine allgemeine SIMD-Unterstützung verfügen.[31]

Seit 2013 ist es nun auf für Webprogramme möglich von SIMD Gebrauch zu machen. Für die von *Google* entwickelte Programmiersprache **Dart** existiert seitdem eine Schnittstelle, die es erlaubt nahezu direkt auf die SIMD-Register zuzugreifen und mit diesen zu arbeiten. Von bis zu 400%igen Beschleunigungen bei speziellen Anwendungen ist die Rede.[32] Auch **JavaScript** soll demnächst durch SIMD-Befehle erweitert werden; Firefox verfügt bereits testweise über den entsprechenden Support.[33]

Wichtige Anwendungsbereiche finden sich heute hauptsächlich im Multimedia-Sektor, aber auch im HPC und wissenschaftlichen Rechnen hat sich SIMD bereits vor langer Zeit bewährt.

F. Vor- und Nachteile

1) Vorteile: Hauptsächlich Multimedia-Anwendungen wie Bild-, Video- und Audiobearbeitung können von SIMD-Rechnern großen Nutzen ziehen, da dort oft Anwendungsfälle auftreten, bei denen die ein und die selbe Operation auf viele Datenpunkte angewandt werden muss. Der Befehl selbst, sowie der zugehörigen Daten können oft (je nach Speicherausrichtung) jeweils mittels eines einzigen Befehls geladen werden. Die Ausführung des Befehls erfolgt dann auf allen Datensätzen zur exakt selben Zeit.

2) Nachteile: Nicht alle Algorithmen sind jedoch ohne Weiteres vektorisierbar. So sind beispielsweise stark verzweigte Kontrollflüsse (*if-else*-Konstrukte) im Gegensatz zu bspw. *for*-Schleifen denkbar ungeeignet für SIMD-Rechner. Auch wenn hier aktuell viel geforscht wird, erfolgt eine performante Vektorisierung zudem immer noch in zu wenigen Fällen automatisch vom Compiler. Oft muss vom Programmierer selbst am Assemblercode nachgeholfen werden. Wie bereits erwähnt, spielt auch die Ausrichtung der Vektoren im Speicher eine große Rolle. Dies ist oft bei verschiedenen Rechnerarchitekturen unterschiedlich gelöst und erfordert somit eine ständige Anpassung des Programmierers an die aktuellen Gegebenheiten.

Literatur

[1] Jon Stokes, *Pipelining: An Overview (Part I)*, http://arstechnica.com/features/2004/09/pipelining-1/, Ars Technica, 2004.

[2] Daniel L. Slotnick, W. Carl Borck, Robert C. McReynolds, *The Solomon Computer*, http://www.computer.org/csdl/proceedings/afips/1962/5061/00/50610097.pdf.

[3] W. J. Bouknight, Stewart A. Denenberg, *The Illiac IV System*, http://research.microsoft.com/en-us/um/people/gbell/Computer_Structures_Principles_and_Examples/csp0322.htm.

[4] *The TI ASC: A Highly Modular and Flexible Super Computer Architecture*, http://research.microsoft.com/en-us/um/people/gbell/Computer_Structures_Principles_and_Examples/csp0769.htm.

[5] *The CDC Star-100*, http://homepages.inf.ed.ac.uk/cgi/rni/comp-arch.pl?Vect/star100.html,Vect/star100cpu-f.html,Vect/menu-cyb.html.

[6] *Cray History*, http://www.cray.com/About/History.aspx.

[7] Jeff Bauer, *A History of Supercomputing at Florida State University*, http://www.ed-thelen.org/comp-hist/super-users-view.html.

[8] *Die Cray 1 - Architektur eines Supercomputers*, http://www.bernd-leitenberger.de/cray-1.shtml.

[9] Gregory V. Wilson, *The History of the Development of Parallel Computing*, http://ei.cs.vt.edu/~history/Parallel.html.

[10] Stephen Shankland, *IBM supercomputing goes retro*, http://archive.today/20130119131859/http://news.com.com/IBM+supercomputing+goes+retro/2100-1010_3-5425551.html#selection-993.1-993.30.

[11] *Sublist Supercomputer Generator*, http://www.top500.org/statistics/sublist/.

[12] Dean Michael Blasco Ancajas, *Trends in Supercomputing*, http://www-inst.eecs.berkeley.edu/~n252/su06/dean_trends.pdf.

[13] Michael Flynn, *Some Computer Organizations and Their Effectiveness*, IEEE Trans. Comput., Band C-21, Seiten 948–960, 2000.

[14] Jon Stokes, *SIMD architectures*, http://arstechnica.com/features/2000/03/simd/, Ars Technica, 2000.

[15] Lawson H.W., *Parallel Processing in Industrial Real-Time Applications*, Prentice-Hall, 1992.

[16] Bertil Svensson, *SIMD Architectures*, http://www.hh.se/download/18.70cf2e49129168da0158000103477/1341267676514/SIMD+Architectures.pdf.

[17] David A. Patterson, *Computer Organization and Design: the Hardware/Software Interface*, 4. Ausgabe - Englisch, Morgan Kaufmann Publishers, 2013, Seiten 509-511.

[18] *SIMD Technology on the Desktop*, http://users.ece.gatech.edu/~linda/6100/Lectures/lec-MMX-SSE-3DNow.PDF.

[19] *AltiVec/SSE Migration Guide*, https://developer.apple.com/legacy/library/documentation/Performance/Conceptual/Accelerate_sse_migration/Accelerate_sse_migration.pdf.

[20] Uwe Harms, *Supercomputing-Rangliste und HPC-Trends*, http://www.tecchannel.de/server/extra/402130/supercomputing_die_neue_top500_liste/.

[21] James Reinders, *AVX-512 instructions*, https://software.intel.com/en-us/blogs/2013/avx-512-instructions.

[22] *Intel® Xeon Phi™ Coprocessor - the Architecture*, https://software.intel.com/de-de/articles/intel-xeon-phi-coprocessor-codename-knights-corner.

[23] *Nvidia Tesla, HPC*, http://www.nvidia.de/object/tesla-high-performance-computing-de.html.

[24] *November 2013 Supercomputer List*, http://top500.org/lists/2013/11/.

[25] Christian Kirbach, *SSL-Beschleunigung mit modernen 3D-Grafikkarten*, Diplomarbeit, Seiten 19-21, 2008.

[26] *Cell Broadband Engine Programming Handbook Including the PowerXCell 8i Processor*, https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/7A77CCDF14FE70D5852575CA0074E8ED.

[27] *PlayStations power Air Force supercomputer*, http://www.cnet.com/news/playstations-power-air-force-supercomputer/.

[28] *25C3: Hackers completely break SSL using 200 PS3s*, http://hackaday.com/2008/12/30/25c3-hackers-completely-break-ssl-using-200-ps3s/.

[29] *About GPGPU*, http://gpgpu.org/about.

[30] Zhuo Feng, Peng Li, *Multigrid on GPU: Tackling Power Grid Analysis on Parallel SIMT Platforms*, http://dent.cecs.uci.edu/~papers/iccad08/PDFs/Papers/08D.1.pdf.

[31] *General-Purpose SIMD within a register: parallel processing on consumer microprocessors*, http://aggregate.org/SWAR/Dis/dissertation.pdf.

[32] *Bringing SIMD to the web via Dart*, https://www.dartlang.org/slides/2013/02/Bringing-SIMD-to-the-Web-via-Dart.pdf.

[33] Brendan Eich, https://brendaneich.com/?s=simd.