B. Datenflußrechner

Autor: Markus Möller

- Bezug: Buch „Datenflußrechner“ von Theo Ungerer. Gehört –natürlich nicht so umfangreich wie hier– zu einem Teil der Vorlesung „Rechnerarchitektur“ bei Herrn Rammig.
- Dieser Extrakt entstand als Vorbereitung auf meine Diplomprüfung (Teil: Vertiefungsgebiet). Er faßt einige Themen einfach zusammen und mag etwas unorthodox erscheinen.
- Erstellt auf Apple Macintosh.

1. Einleitung

Mit dem Begriff „Datenfluß“ sind drei Aspekte verknüpft:

1. das *Einmalzuweisungsprinzip* der Datenflußsprachen,
2. die (Maschinen-) Programmdarstellung als *Datenflußgraph* mit einer datengesteuerten Auswertungsstrategie und
3. die *Verarbeitungspipeline* eines Datenflußprozessors, die aus Einheiten für das „Token Matching“, für die Befehlsbereitstellung, die Befehlsausführung und das Erzeugen von neuen „Tokens“ besteht.

Datenflußrechner werden in einer Datenflußsprache programmiert, in der jeder Variablen nur einmal ein Wert zugewiesen werden kann. So ein Datenflußprogramm wird zu einem Datenflußgraphen compiliert, dessen Knoten den Maschinenbefehlen und dessen Kanten den Datenabhängigkeiten zwischen den Befehlen entsprechen.

Die Operanden der Maschinenbefehle werden als Datenpakete („Tokens“) entlang der Kanten übertragen. Die Befehlsausführung geschieht beim Datenflußprinzip datengesteuert, d.h. jeder Befehl wartet für seine Ausführung auf seine Operanden. Es gibt also keinen Befehlszähler. Wenn auf allen benötigten Kanten Tokens vorhanden sind, wird von jedem Eingang ein Token entfernt und ein Resultattoken im Ausgang erzeugt.

Man erwartet vom Datenflußprinzip folgende Vorteile:

- Ausnutzung von Parallelität auf Befehlsebene (kein Programmzähler!),
- dynamische Lastverteilung und

- geringen Aufwand für die Synchronisation paralleler Aktivitäten.

Die ersten Datenflußrechner arbeiteten nach dem „statischen“ Datenflußprinzip. Dieses läßt keine Rekursivität zu und kann Schleifeniterationen und Unterprogrammaufrufe nicht parallel ausführen. Daher nutzt es die vorhandene Parallelität nicht vollständig aus. Mehr als ein Token auf einer Kante ist beim Prinzip des statischen Datenflusses verboten.

Um mehr Parallelität zu nutzen, wurde das „dynamische“ Datenflußprinzip entwickelt. Hier können voneinander unabhängige Schleifeniterationen und Unterprogrammaufrufe parallel ausgeführt werden. Dazu werden die Tokens um „Tags“ erweitert, die den Kontext (Programmumgebung) der Tokens identifizieren.
Ein Knoten im dynamischen Modell ist schaltbereit, wenn an jeder Eingangskante Token mit identischen Tags vorhanden sind.

Einelementige Operationen können sofort durchgeführt werden. Bei zweistelligen muß festgestellt werden, ob beide Operanden verfügbar sind. Dieses Token Matching wird von einer Vergleichseinheit, Matching Unit, durchgeführt. Bis vor kurzem wurde dazu ein in Hardware implementiertes Hash-Verfahren realisiert, das durch das bessere Verfahren des „direkt adressierbaren Token-Speichers“ abgelöst wurde.

Datenflußrechner tragen im Vergleich zu von-Neumann-Multiprozessorsystemen zur Lösung der Probleme der Programmierbarkeit, der Speicherlatenz und der Synchronisation bei:

- Leichte Programmierbarkeit:
  In Datenflußsprachen ist Parallelität implizit vorhanden und muß nicht vom Programmierer ausgedrückt werden.
- Speicherlatenz:
  Da jeder Befehl in der Pipeline eines Datenflußprozessors einen anderen Kontext (Tag) haben kann, können Verzögerungen durch Speicherzugriffe toleriert werden. Die Daten können in beliebiger Reihenfolge aus einem entfernten Speicher geliefert werden. Wer zuerst ankommt rechnet zuerst.
  Ein Prozessor eines von-Neumann-Multiprozessorsystems arbeitet jedoch einen sequentiellen Code ab, der aus einem Kontext stammt.
- Synchronisation:
  Die Synchronisation verschiedener Kontrollfäden geschieht auf Maschinenebene durch das Token-Matching-Prinzip, wobei jeder Maschinenbefehl darauf wartet, daß alle seine Operanden produziert sind, bevor er ausgeführt wird. Also nicht durch den Programmierer,

der dann verschiedene Kontrollfäden auf verschiedene Prozessoren verteilen muß.

Beim dynamischen Datenflußrechner existiert ein Tokenpuffer (Token Queue), der die Tokens enthält, die auf ihren Tag-Partner warten.

Beim statischen werden die Tokens von einer **Schalteinheit** in ihren Aktivitätseintrag[1] (Operationscode, Operandenplätze, Zielfelder für Ergebnis) im **Aktivitätsspeicher** geschrieben, da jeder Befehl ja nur einmal aktiviert sein kann, und falls der Befehl dann vollständig ist, der Eintrag in den Grundzustand zurückgesetzt und der Inhalt an eine **Verarbeitungseinheit** geschickt.

Beim dynamischen werden die eingehenden Token von einer **Vergleichseinheit** auf vorhandene Partner im Tokenpuffer überprüft und dann entweder beide an die Befehlsbereitstellungseinheit weitergereicht, wenn ihr Partner im Tokenpuffer ist oder sie keinen brauchen, oder im **Tokenpuffer** abgelegt, wenn ihr Partner nicht vorhanden ist. Die **Befehlsbereitstellungseinheit** holt den zugehörigen Befehlseintrag aus dem **Programmspeicher** und fügt den Operationscode und die Zieladressen zum Token-Paar hinzu. Dieses vollständige Befehlspaket wird an irgendeine **Funktionseinheit** geschickt.

Der Tokenpuffer in Kombination mit dem Programmspeicher des dynamischen Systems entspricht im Prinzip dem Aktivitätsspeicher des statischen Systems mit dem Unterschied, daß im Tokenpuffer Tokens für identische Operationen aus verschiedenen Kontexten möglich sind während im Aktivitätsspeicher jede Operation (Knoten) nur einmal vorkommt. Das macht das dynamische Prinzip paralleler.

Bei sequentiellen Befehlsfolgen erweist sich das Token-Matching-Prinzip (oder allgemein auch das Datenflußprinzip) gegenüber dem Befehlszählerprinzip der von-Neumann-Rechner als ineffizient. Da die zeitliche Reihenfolge, in der die Befehle eines Datenflußprogramms von der Pipeline eines Datenflußprozessors ausgeführt werden, nicht festlegbar ist, können Register als Zwischenspeicher für Operanden nicht genutzt werden. Die Operanden (Token) fließen eventuell also erstmal durch ein Verbindungsnetz zu einem anderen Prozessor. Ein Befehl kann erst dann ausgeführt werden, wenn sein Vorgängerbefehl die gesamte Datenflußpipeline durchlaufen hat.

1. Entspricht einem Maschinenbefehl bzw. Datenflußknoten

Abhilfe wird geschaffen, in dem mit dem „Multithreaded Dataflow“ Verfahren sequentielle Befehlsfolgen direkt aufeinanderfolgend unter Verwendung von Registern ausführt werden. Man erspart sich dadurch, die Resultattokens über die Vergleichseinheit und die Befehlsbereitstellungseinheit schließlich wieder zu einer Ausführungseinheit zu senden, und behält das Ergebnis gleich dort.

Andersherum wird auch bei von-Neumann-Prozessoren der Vorteil des schnellen Kontextwechsels gesucht, um Wartezeiten bei Speicherzugriffen zu überbrücken. Hier werden statt eines einzigen gleich mehrere Kontrollfäden (Threads) auf einen Prozessor geladen, wobei jedem Kontrollfaden ein eigener Registersatz zugeordnet ist. Bei fehlgeschlagenen Synchronisationsversuchen oder Wartezeiten wird auf einen anderen Registersatz gewechselt und damit ein Kontextwechsel herbeigeführt. Dieser geht schnell, da das Abspeichern der Registerinhalte, das bei konventionellen von-Neumann-Prozessoren notwendig ist, entfällt.

Die letzten beiden Absätze beschrieben Hybridarchitekturen aus beiden Typen. Der Vorteil des Datenfluß ist halt, daß jeder verarbeitete Befehl aus einem anderen Kontext stammen kann. Er wird aktiviert, wenn seine Operanden ankommen.

2. Grundlagen des Datenflußprinzips

2.1 Kontrollfluß-, Reduktions- und Datenflußprinzip

Dies sind die drei grundlegenden Operationsprinzipien für Rechner.

Das Kontrollflußprinzip kann mit dem von-Neumann-Prinzip gleichgesetzt werden, das durch die sequentielle Befehlsfortschaltung durch einen Befehlszähler charakterisiert ist. Der Befehlszähler verweist immer auf die Speicheradresse des in der sequentiellen Ausführungsreihenfolge nächsten Maschinenbefehls.

Bei parallelen Kontrollflußprinzipien können Befehlsfolgen, die nach dem von-Neumann-Prinzip ausgeführt werden, durch explizite parallele Kontrollkonstrukte (z.B. fork…join) verknüpft sein und parallel zueinander ausgeführt werden.

Beim Datenflußprinzip ist die Ablaufsteuerung gänzlich anders organisiert. Einzig die Verfügbarkeit der Operanden löst die Ausführung einer Maschinenoperation auf diesen Operanden aus. Die Resultate können dann wieder zur Ausführbarkeit anderer Operationen führen. Deshalb wird dieses Operationsprinzip auch als datengesteuert oder datengetrieben bezeichnet.

Beim Reduktionsprinzip wird ein Maschinenbefehl ausgeführt, wenn einer seiner Ausgabeoperanden von einem anfordernden Maschinenbefehl benötigt wird. Nach der Ausführung der Maschinenoperation wird die Kontrolle wieder an den anfordernden Maschinenbefehl zurückgegeben.

Die Anforderung basiert auf dem Erkennen reduzierbarer Ausdrücke und der Reduktion dieser Ausdrücke.Bei der Reduktion wird ein Ausdruck durch sukzessive Anwendung von Reduktionsregeln (Ersetzungsregeln) umgeformt. Dabei werden reduzierbare Ausdrücke durch gleichwertige Ausdrücke ersetzt, bis das Resultat, d.h. der maximal vereinfachte Ausdruck, erreicht ist.

Als Reduktionsebene wird im folgenden derjenige Schnitt durch den Berechnunsbaum bezeichnet, der durch einmalige Reduktion aller Teilausdrücke eines anderen Schnitts entsteht. Das Ersetzen eines Teilausdrucks geschieht rekursiv, wobei von einer Reduktionsebene zur nächsten geschritten wird, bis die Elementaroperationen erreicht sind, die dann ausgeführt werden. Darauf werden die Ergebnisse der nächst-höheren Reduktionsebene zurückgemeldet.

Im Prinzip besteht ein Knoten des Reduktionsbaumes aus zwei Operanden, die Variablen aus der nächsten Ebene oder Zahlen sind, und einem Operator. Die Blätter sind in jedem Fall Zahlen, die eine Variable füllen.

Es gibt zwei verschiedene Reduktionsformen:

- Textersetzung
  Hier wird bei jeder Anforderung eines Ausdrucks (also Variablen) eine Kopie ihrer Definition (kann auch wieder ein Ausdruck oder Zahl sein) einer tieferen Reduktionsebene in den anfordernden Ausdruck kopiert. Falls die Antwort auch ein Ausdruck ist, wird dieser auch angefordert usw.
- Graphreduktion
  Hier wird mit einem Zeiger jeweils der anfordernde mit dem angeforderten Ausdruck verbunden. Im Gegensatz zur Textersetzung muß jeder Ausdruck nur einmal berechnet werden, der mehrmals angefor-

dert wird, da ab dem zweiten Mal direkt auf den berechneten Wert zurückgegriffen werden kann.

2.2 Datenflußsprachen

Die Reihenfolge der Anweisungen in einem Datenflußprogramm spielt innerhalb eines Anweisungsblocks (Funktion, Schleifenkörper, if-, then- oder else-Zweig) keine Rolle. Im Gegensatz zu imperativen, parallelen Sprachen, bei denen die Parallelität durch explizite Parallelkonstrukte ausgedrückt wird, sind Datenflußsprachen implizit parallel. Manchmal wäre es jedoch effizienter, wenn der Programmierer die Ausführung beeinflussen könnte.

In Datenflußsprachen gilt das Prinzip der Einmalzuweisung, d.h. jeder Variablen in einem Datenflußprogramm kann nur einmal ein Wert zugewiesen werden.

Datenflußsprachen haben viel mit funktionalen Sprachen gemeinsam und werden mit ihnen zu den sog. *applikativen Sprachen* zusammengefaßt. Der Begriff kommt vom grundlegenden Konzept der Anwendung (Application) von Funktionen auf Argumente.

Applikative Sprachen sind *seiteneffektfrei,* d.h. im Rumpf einer Funktion können keine Wertzuweisungen zu Variablen außerhalb des Funktionsrumpfes vorkommen.

Die Ausdrücke applikativer Programmiersprachen können von außen nach innen oder von innen nach außen ausgewertet werden. Im ersten Fall werden Funktionsaufrufe ausgewertet, wenn ihre Resultate als Argumente von anderen Funktionen benötigt werden. *Bedarfsgesteuerte Auswertung*, Reduktionsprinzip. Von innen nach außen entspricht der d*atengesteuerten Auswertungsstrategie* des Datenflußprinzips.

Eine Funktion heißt *strikt,* wenn beim Auftreten eines undefinierten Ausdrucks oder einer nicht terminierenden Rekursion an einer Argumentstelle die gesamte Funktionsauswertung den Wert ‘undefiniert’ erhält, andernfalls heißt die Funktion *nicht-strikt*.

Eine Semantik, bei der alle Funktionen strikt definiert sind, heißt *strikte* Semantik, das Gegenteil dazu *nicht-strikte* Semantik.

Wenn bei der Anwendung einer Funktion stets ihre sämtlichen Argumente vollständig ausgewertet werden, spricht man von einer *call-by-value-Semantik*, andernfalls von einer *call-by-name-* oder einer *call-by-need-Semantik*.

Man unterscheidet drei Auswertungsstrategien: *eager*, *lazy* und *lenient*. Bei einer strikten Semantik wird meist *eager* ausgewertet, also so parallel wie möglich, auch wenn dies zur Erzielung des Resultats „unnötige“ Auswertungen bewirkt.

Die Auswertungsstrategie bei einer nicht-strikten Semantik heißt *lazy*, falls nur die Ausdrücke ausgewertet werden, die zur Berechnung des Resultats notwendig sind. Lazy-Auswertung führt zu einer bedarfsgesteuerten Auswertungsstrategie.

Bei einer nicht-strikten Semantik heißt die Auswertungsstrategie *lenient* (milde, nachsichtig), wenn bei einem Konditional erst das Prädikat und danach der ausgewählte Zweig ausgeführt wird. Alle anderen Funktionen werden eager ausgewertet.

Ein weiteres Unterscheidungskriterium einer Sprache: Man spricht von *strenger Typprüfung*, falls nicht übereinstimmende Datentypen immer entdeckt werden, von *statisch getypt*, falls die Übereinstimmung von Datentypen zur Compilezeit geprüft werden kann, und von *dynamisch getypt*, falls die Typenprüfung erst zur Laufzeit möglich ist.

Man spricht von *Polymorphie*, wenn eine Funktion so definiert ist, daß sie einheitlich die gleiche Operation auf verschiedenen Datentypen ausführt, und von *überladen*, wenn eine Funktion je nach Datentyp ihrer Argumente unterschiedliche Operationen bezeichnet.

2.3 Datenflußgraphen und Berechnungsschemata

Bei der Compilation von Datenflußprogrammen wird zunächst in eine Zwischensprache übersetzt und dann der jeweilige Maschinencode erzeugt. Die Darstellung in der Zwischensprache läßt sich durch einen Datenflußgraphen zeigen. Dieser ist gerichtet. Seine Knoten repräsentieren Befehle und seine Kanten Datenabhängigkeiten. Operanden für Befehle werden entlang der Kanten in Form von Datenpaketen, *Tokens* genannt, übertragen. Die Ausführung eines Befehls wird *Schalten* genannt. Für alle Datenflußrechner gilt:

> Ein Knoten ist schaltbereit, sobald auf allen Eingangskanten Tokens vorhanden sind.

Beim Schalten wird von jeder Eingangskante ein Token entfernt und auf jeder Ausgangskante ein Resultattoken abgelegt.

Ein Datenflußgraph heißt *sicher*, wenn beim Ablauf eines Programms keine Kante zu einem Zeitpunkt mehr als ein Token enthalten kann.

Ein *Branch-Knoten* stellt einen bedingten Sprungbefehl in einem Kontrollflußprogramm dar. In Abhängigkeit von einem Steuerkanal wird das Eingabetoken unverändert an einen der beiden Ausgabekanäle weitergereicht, die mit ‘wahr’ bzw. ‘falsch’ benannt sind.

Der *Merge-Knoten* ist meist mit einer nicht-strikten Schaltregel definiert. Er schaltet sein Eingangstoken auf den Ausgang, sobald einer seiner Eingangskanäle ein Token erhält. Sein deterministisches Pendant hat eine zusätzlichen Steuerkanal, der angibt, aus welchem Eingabekanal ein Token genommen wird.

Bei sogenannten *Verbund-Branch-* oder *Verbund-Merge-Knoten* haben die Knoten mehr als jeweils einen Eingabekanal. Der Knoten schaltet, wenn an allen vervielfachten Eingabekanälen Tokens anliegen. Diese Knoten werden für Schleifen benutzt. Sie wirken als Sperre, weshalb diese Methode, Schleifen zu implementieren, als *Sperrmethode* bezeichnet wird. Da hierdurch Schleifeniterationen sequentiell nacheinander ausgeführt werden, für Parallelverarbeitung unattraktiv.

Die *Rückkopplungsmethode* kann auch Schleifen realisieren und verschiedene Iterationen teilweise überlappt verarbeiten unter Inkaufnahme von doppeltem Tokenverkehr.

Eine dritte Methode ist die *Code-Copying-Methode*, bei der für jede Iteration der Schleife oder eines Unterprogrammaufrufs eine Kopie des Datenflußteilgraphen angelegt wird. Damit volle Parallelität, aber hoher Speicherverbrauch.

Eine speichereffizientere Methode ist die *Tagged-Token-Methode*. Der Teilgraph wird gemeinsam gespeichert und die Tokens mit zusätzlichen Tags versehen, um die Iterationen zu unterscheiden. Kanten als Tokenbehälter. Ein Tag enthält ID des Unterprogrammaufrufs (Kontextnummer), der Schleifeniteration (Iterationsnummer) und des Befehls, dessen Operand das Token transportiert. Knoten schaltet, wenn Tokens mit identischen Tags an allen Eingängen vorliegen. Sicherheit bedeu-

tet hier, daß keine Kante Tokens mit identischen Tags hat. Hiermit vollständige Nutzung von Parallelität zwischen Schleifeniterationen oder Unterprogrammaufrufen. Beim Unterprogrammaufruf wird ein neuer Tagbereich mit einer neuen Kontextnummer eingerichtet.

Sperrmethode und Rückkopplungsmethode kennzeichnen *statische Datenflußrechner*, Kopier- und Tagged-Token-Methode *dynamische* Architekturen.

2.4 Grundstrukturen der Datenflußrechner

In statischen Datenflußrechnern werden die Knoten eines Datenflußgraphen in ihrer Maschinenrepräsentation als Einträge in einer Aktivitätstabelle beschrieben. So ein Aktivitätseintrag enthält den Opcode, Speicherplätze für die Operandenwerte und Zielverweise.

Die Operationen auf den Aktivitätstabellen führt die Schalteinheit durch. Die Befehle werden von der Funktionseinheit (Verarbeitungseinheit) ausgeführt.

Die Verbindungen zwischen den beiden Einheiten sind durch Pufferspeicher realisiert.

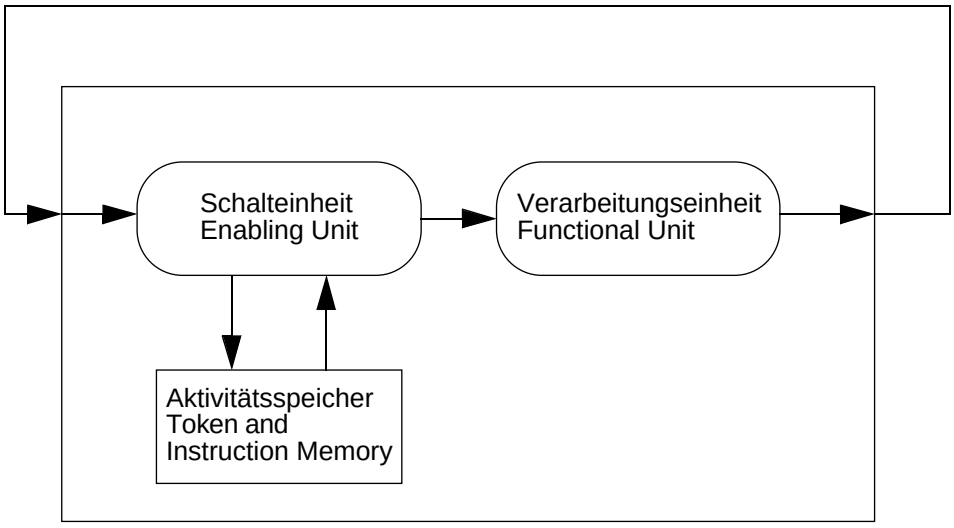

ABBILDUNG 1: Verarbeitungselement eines statischen Datenflußrechners

Um den Code reentrant[2] zu halten, ist bei dynamischen Datenflußrechnern die gemeinsame Speicherung von Tokens mit dem Opcode und den Zielverweisen in Aktivitätseinträgen nicht sinnvoll.

2. Reentrant bedeutet, daß derselbe Programmcode mehrfach parallel ausgeführt werden kann, ohne daß der kopiert werden muß.

Wartende Tokens werden deshalb von den Datenflußknoten getrennt gespeichert. Tokens werden im Token-Speicher abgelegt und von der Vergleichseinheit manipuliert. Der Code wird im Befehlsspeicher untergebracht, auf den die Befehlsbereitstellungseinheit zugreift. Diese Unterteilung der Schalteinheit in zwei verschiedene Einheiten erlaubt eine überlappt parallele Arbeit beider Einheiten in einer Pipeline.

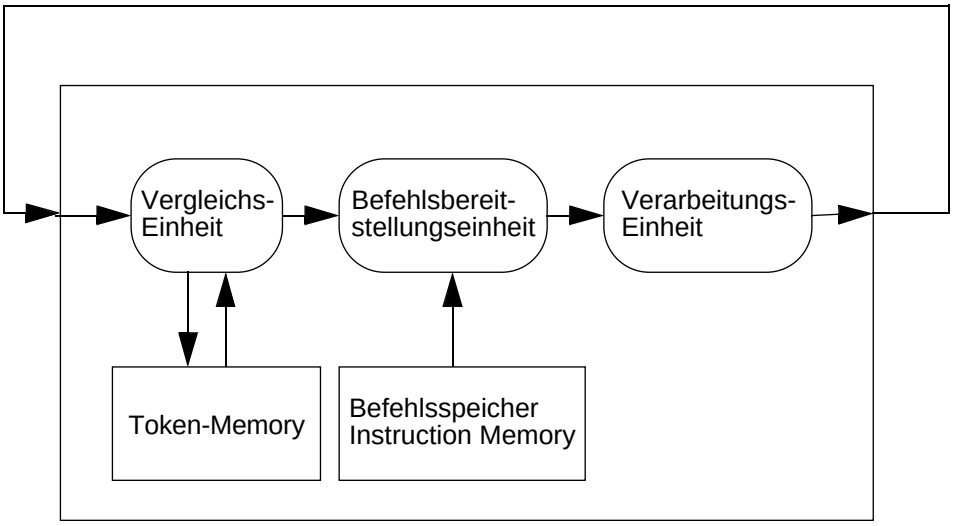

ABBILDUNG 2: Verarbeitungselement eines dynamischen Datenflußrechners

Die Suche nach einem Partnertoken erfordert beim dynamischen Datenflußprinzip einen assoziativen (inhaltsbezogenen) Speicherzugriff über den Tag. Um dies zu umgehen wird entweder eine Hashfunktion oder ein Adressierungsverfahren über Offset- und Basisadressen von Aktivierungsrahmen verwendet. Vergleichseinheit und Befehlsbereitstellungseinheit bilden die Schaltstufe und die Funktionseinheit die Ausführungsstufe.

Je nach Körnigkeit der Aktivität, für die eine Vergleichsoperation von der Vergleichseinheit durchgeführt wird, kann man unterscheiden:

- feinkörnige Datenflußarchitekturen: jede Aktivierung betrifft nur einen einzigen Befehl geringer Komplexität,

- Datenflußarchitekturen mit komplexen Maschinenoperationen: die Aktivitäten betreffen einzelne komplexe Befehle, z.B. Vektorbefehle,
- Multithreaded-Datenflußarchitekturen: eine Befehlsfolge wird nach der Aktivierung des ersten Befehls ausgeführt,
- Large-Grain[3]-Datenflußarchitekturen: eine Aktivität entspricht einer ganzen Befehlsfolge.

Bei den feinkörnigen Datenflußrechnern unterscheidet man prinzipiell drei Architekturen:

- Die heute übliche One-Level Dataflow Machine bzw. Datenflußmultiprozessor: Mehrere Verarbeitungselemente arbeiten parallel zueinander und kommunizieren durch Tokens über ein Verbindungsnetz.
- Die Two-Level Dataflow Machine verfügt in jedem Verarbeitungselement über mehrere parallel geschaltete Funktionseinheiten. Wird nicht mehr angewandt, da in der Verarbeitungspipeline die Vergleichseinheit und nicht die Funktionseinheit der Flaschenhals ist.
- Die Two-Stage Dataflow Machine trennt die Verarbeitungseinheiten in Schalteinheiten und Funktionseinheiten, die über zwei unidirektionale Verbindungsnetze kommunizieren. Wegen hohem Hardwareaufwand und Verzögerung des Tokenflusses durch die Kommunikationsnetze nicht praktikabel.

2.5 Klassifikation der Datenflußrechner

Das wesentliche Unterscheidungskriterium für (feinkörnige) Datenflußrechner ist statisch/dynamisch. Ferner kann man zwischen direkter Kommunikation und Paketkommunikation unterscheiden. Bei den dynamischen zwischen Kopier- und Tagged-Token-Methode.

Bei der Direktkommunikation werden benachbarte Knoten eines Datenflußgraphen auf dasselbe Verarbeitungselement oder auf benachbarte abgebildet. Falls FIFO-Pufferung verwendet wird, können sogar statische Datenflußgraphen ohne Rückkopplungsmethode überlappt parallel und determiniert ausgeführt werden.

Üblicher ist bei Datenflußmultiprozessoren das Paketverfahren, bei dem die Tokens gemäß ihrer Verarbeitungselementadresse vom Kommunikationsnetz zu den Verarbeitungselementen transportiert werden. Der Tag wird dafür um die Verarbeitungselementadresse erweitert.

3. grob-körnig

Heute wird nur noch das dynamische Prinzip mit Tagged-Token, Paketkommunikation und Multiprozessorstruktur weiterverfolgt.

2.6 Erweiterungen feinkörniger Datenflußrechner

Bei feinkörnigen Datenflußarchitekturen bezeichnet jeder Befehl eine Maschinenoperation geringer Komplexität. Probleme bei diesem Prinzip:

- Bei dyadischen Befehlen führt die Ankunft des ersten Token zu einer Blase im Befehlsstrom der Pipeline der nachfolgenden Ausführungsstufe.
- Falls der Graph zu einer sequentiellen Befehlsfolge degeneriert ist oder nur gering parallel ist, bedeutet der Tokentransport durch die zirkuläre Pipeline eine Verzögerung für die Ausführung der Befehlsfolge, da ein Resultattoken die gesamte zirkuläre Pipeline durchlaufen muß.
- Wegen regelmäßiger Kontextwechsel wird auf Register verzichtet, somit auf optimierte Zugriffszeiten. Pipelineblasen könnten verhindert werden und die Gesamtzahl der erzeugten Tokens.

Abhilfe schafft die Multithreaded-Architektur, bei der sequentielle Befehlsfolgen aufeinanderfolgend an die Ausführungsstufe weitergereicht werden, nachdem nur für den ersten Befehl eine Vergleichsoperation stattfand. Hier können für den einen Kontrollfaden Register verwendet werden. Man unterscheidet dabei das direkte Wiedereinfüttern von Tokens und die direkt aufeinanderfolgende Ausführung der Befehlsfolge.

Die Synchronisation auf Befehlsebene läßt sich auch durch komplexe Maschinenbefehle wie Vektorbefehle vereinfachen. Strukturierte Daten werden dabei einmalig und nicht elementweise angesprochen. Solche Befehle können mehrere geschachtelte Schleifen ersetzen und vermindern den Tokenverkehr.

Ähnlich dem Multithreadedprinzip werden bei den Large-Grain-Architekturen Folgen von Befehlen zu sequentiellen Codeblöcken zusammengefaßt. diese werden aber von *einem* Knoten (einem „Makrodatenflußknoten“) im Graphen repräsentiert. Large-Grain-Architekturen aktivieren solche Makroknoten nach dem Datenflußprinzip und führen dann die repräsentierte Befehlsfolge nach dem von-Neumann-Prinzip aus.

2.7 Anwendungen der Datenflußrechner

Datenflußrechner sind immer dann anderen überlegen, wenn es darum geht, Parallelität auf Befehlsebene erst zur Laufzeit zu erkennen.

Die Verwendung des Datenflußprinzips gründet einerseits auf der relativ zum Programmieraufwand einfachen Erzeugung von Parallelität, die von mehreren Prozessoren genutzt werden kann, und andererseits auf der Unempfindlichkeit gegen Speicherlatenzzeiten.

Die potentiell größte Anwendungsmöglichkeit neben Spezialanwendungen entsteht durch die Fähigkeit zum schnellen Kontextwechsel, der es erlaubt, Speicherlatenzzeiten und Wartezeiten auf eine Prozeßsynchronisation besser als durch konventionelle Multiprozessoren zu überbrücken.

3. Parallelitätsebenen und Parallelarbeitstechniken

3.1 Ebenen der Parallelität

Die Körnigkeit (Grain) bemißt sich nach der Anzahl der Befehle in einer sequentiellen Befehlsfolge. Hinsichtlich der Körnigkeit paralleler Programme werden fünf Parallelitätsebenen unterschieden:

- Programmebene (Jobebene): Parallele Verarbeitung verschiedener Programme, die vollständig voneinander unabhängig sind.
- Taskebene (Prozeßebene): Programm wird in parallel zu bearbeitende Tasks zerlegt, die aus einigen tausend sequentiell ausgeführten Befehlen bestehen. UNIX-Prozesse z.B.
- Blockebene: Anweisungsblöcke oder leichtgewichtige Prozesse. Mehrere Einzelanweisungen bis zu einigen hundert zu Anweisungsblock zusammengefaßt, daß verschiedene Anweisungsblöcke parallel ausgeführt werden können. Innerer oder äußere Schleifen.
- Anweisungsebene (Befehlsebene): Elementare Anweisungen parallel zueinander ausgeführt. Wird durch Datenflußsprache direkt ausgedrückt.
- Suboperationsebene: Elementare Anweisung wird durch den Compiler oder in der Maschine in Suboperationen aufgebrochen, die parallel ausgeführt werden. Vektoroperationen.

3.2 Techniken der Parallelarbeit durch Prozessorkopplung

3.2.1 Rechnernetze

Rechnernetz ist ein Kommunikationsnetz, das eine Anzahl unabhängig voneinander arbeitender Rechner verbindet. 10 MBit pro Sekunde. Synchronisationsaufwand ist erheblich. Parallelarbeit auf lokalen Rechnernetzen ist deshalb z.Z. am ehesten auf Taskebene für seht große Tasks effizient einsetzbar.

3.2.2 Nachrichtengekoppelte Multiprozessoren

Als Multiprozessorsysteme (Multiprozessoren) bezeichnet man Rechner, die in die MIMD-Klasse fallen. Im Unterschied zu Feldrechnern (SIMD) bei denen ein „Feld“ von Prozessoren gleichzeitig die gleiche Operation auf verschiedenen Datenobjekten ausführt, können in Multiprozessoren die einzelnen Prozessoren unabhängig voneinander programmiert werden.

Bei den nachrichtengekoppelten Systemen gibt es keine gemeinsamen Speicher- oder Adreßbereiche. Kommunikation durch Nachrichtenaustausch über Verbindungsnetz. Nur lokale Speicher für die Prozessoren. 20-50 MBit pro Sekunde. Parallelität in effizient auf Programm- oder Taskebene nutzbar.

Bei den nachrichtengekoppelten Systemen der zweiten Generation wird die Kommunikation nichtbenachbarter Knoten nicht mehr durch Software, sondern durch spezielle Kommunikationshardware organisiert.

Datenflußrechner sind Multiprozessoren, deren Knoten Datenflußprozessoren sind. Bei heutigen Systemen ist vorwiegend eine Speicherkopplung anzutreffen. Aber auch bei Datenflußmultiprozessoren ohne gemeinsamen Speicher geschieht die Kommunikation zwischen den Datenflußprozessoren nicht über Austausch von Nachrichten, sondern von Tokens.

3.2.3 Speichergekoppelte Multiprozessoren

Bei den speichergekoppelten Systemen sind Prozessoren über gemeinsam zugreifbare Speicherbereiche gekoppelt. Meist existiert ein einziger globaler Speicher. Einfacher programmierbar als nachrichtengekoppelte. Nutzbare Parallelität von der Programmebene bis zur Blockebene.

Die meisten Datenflußrechner sind speichergekoppelte Multiprozessoren, wobei der globale Speicher aus Strukturspeicher-Modulen zur Speicherung komplexer Datenstrukturen besteht. Beispiel: I-Strukturen. Die Strukturspeicher-Module kommunizieren mit den Datenflußprozessoren durch Austausch von Tokens und sich daher eher aktiver als bei konventionellen Systemen.

3.2.4 Virtual-Shared-Memory-Architekturen

Kombination aus speicher- und nachrichtengekoppelten Systemen, bei denen in einer für die Software transparenten Weise die Sicht eines speichergekoppelten Multiprozessorsystems auf einem nachrichtengekoppelten Multiprozessorsystem implementiert wird. Kombination von leichter Skalierbarkeit nachrichtengekoppelter Systeme mit einfacher Programmierbarkeit speichergekoppelter Systeme.

3.2.5 Typische Probleme von Multiprozessoren

Hauptproblem auf Softwareebene ist die schwierige Programmierbarkeit. Auf Hardwareebene Speicherlatenz und Synchronisation paralleler Kontrollfäden.

Datenflußprinzip: Durch implizite Parallelität leichte Programmierbarkeit. Durch Matching-Operation Toleranz beliebiger Verzögerungen und Reihenfolge der Daten. Synchronisation paralleler Kontrollfäden durch Matching-Operation erzwungen.

3.3 Techniken der Parallelarbeit in der Prozessorarchitektur

3.3.1 Befehlspipelining und Superpipelining

Pipelining ist die Zerlegung einer Maschinenoperation in mehrere Phasen oder Suboperationen, die dann von hintereinander geschalteten Verarbeitungseinheiten taktsynchron bearbeitet werden.

Beim Superpipelining werden die Stufen einer Befehlspipeline in noch feinere Pipelinestufen unterteilt und diese mit einer hohen Taktfrequenz, z.B. einem schnelleren internen Takt, ausgeführt.

Ein dem Befehlspipelining verwandtes Verfahren wird in Form der Token-Pipeline bei allen dynamischen Datenflußprozessoren verwendet. Pipeline-Konflikte (Aus-

führung von Befehl dauert länger als ein Takt, Registerkonflikte, Sprünge) sind hier ausgeschlossen, da die Befehle in der Pipeline aus verschiedenen Kontexten stammen.

3.3.2 Superskalare Prozessoren und VLIW-Maschinen

Ein superskalarer Prozessor kann mehrere Befehle pro Taktzyklus einspeisen, da er mehrere nebeneinander angeordnete Befehlspipelines und mehrere Funktionseinheiten für die Befehlsausführung hat. Umfang der Parallelarbeit vergleichbar zu Superpipeline. Parallelität auf Anweisungsebene in beiden Fällen.

LIW-Maschinen bestehen aus einer Anzahl von Funktionseinheiten, die jeweils eine Maschinenoperation taktsynchron ausführen. Alle Funktionseinheiten werden von einem einzigen Maschinenbefehl gesteuert, der mehrere Maschinenoperationen „enthält“. Parallelarbeit auf Maschinenbefehlsebene. Schwierig, wenn ein Teilbefehl Ärger macht. Ist dem Datenflußprinzip überlegen, wenn die Befehlsebenenparallelität zur Compilezeit feststeht und keine störenden Laufzeitereignisse eintreten.

3.4 SIMD-Techniken

3.4.1 Vektorrechnerprinzip

Arithmetische Operationen auf Vektoren von Gleitpunktzahlen in Pipeline-Verfahren ausgeführt. Parallelarbeit sehr effizient auf Suboperationsebene. Grad der nutzbaren Parallelität auf Länge der Pipeline beschränkt.

3.4.2 Prinzip der Datenstrukturarchitektur

Datenobjekte nicht mehr direkt über Speicheradressen sondern durch Anwendung von geeigneten Zugriffsfunktionen manipuliert. Komplexe Datenobjekte als Ganzes adressieren. Vektorrechner nur unvollständiger Schritt in diese Richtung, weil als strukturierte Datenobjekte einzig numerische Vektoren unterstützt werden.

Bei von-Neumann-Architektur dagegen werden die Elemente eines Datenobjekts einzeln adressiert und manipuliert.

Zugriff auf Datenobjekte durch Zugriffsfunktionen mit beliebiger Restrukturierung und Umordnung der Elemente durch Strukturprozessor. Unterschied zu Vektorrechnern, die nur Zugriffe auf Vektoren erlauben, die die Ordnung der Vektorelemente beibehalten.

3.4.3 Feldrechner- und verwandte Architekturprinzipien

Feldrechner haben Feld von regelmäßig verbundenen Verarbeitungselementen, beaufsichtigt von zentraler Steuereinheit, immer gleichzeitig dieselbe Maschinenoperation auf verschiedenen Daten. Massiv parallele Rechner.

Falls Gewicht weniger auf Rechenoperationen als auf Zugriff auf Daten mit bestimmten Eigenschaften, sog. Assoziativrechner: Assoziativspeicher vorhanden, bei dem der Zugriff auf die Daten nicht über Adressen, sondern über die Dateninhalte selbst oder Teile davon geschieht.

Zellulare Systeme: Feld von Verarbeitungseinheiten, verschiedene Operationen, dezentrale Kontrolle.

Systolische Arrays: Feld von Verarbeitungselementen, Ein- und Ausgabe am Rand des Feldes, identische Programme, Datenströme pulsieren mit synchronem Takt durch das Array, keine zentrale Aufsicht.

3.5 Mehr-Ebenen-parallele Rechner

Um eine hohe Verarbeitungsleistung durch umfassende Ausnutzung der im Programm vorhandenen Parallelität zu erreichen, kann ein Rechner Parallelität auf mehreren Ebenen in Parallelarbeit umsetzen.

Heutige Datenflußmultiprozessoren stellen meist Zwei- oder Mehr-Ebenen-parallele Rechner dar. Feinkörnige Parallelität wird für eine überlappende Verarbeitung in der Token-Pipeline eines Datenflußprozessors verwendet, während Block- und Taskebenenparallelität durch Verteilen auf verschiedene Datenflußprozessoren genutzt wird.

4. Statische Datenflußrechner

4.1 Rückkopplungsmethode des statischen Datenflußprinzips

> Ein Knoten ist schaltbereit, sobald auf allen Eingangskanten (einschließlich der Bestätigungskanten) Tokens vorhanden sind.

Das Prinzip des statischen Datenflusses verbietet mehr als ein Token auf einer Kante. Daher werden zusätzliche Bestätigungskanten eingeführt, die dem Lieferknoten anzeigen, das die Datenkante wieder frei ist.

Dadurch ist nur wenig Parallelität möglich, da nur Überlappung von Schleifeniterationen durch den Pipelineeffekt erzielt werden kann. Vollständige Parallelität jedoch nie. Außerdem wird der Tokenverkehr verdoppelt.

Außerdem ist das statische Datenflußprinzip nicht mächtig genug alle Programmkonstrukte moderner Programmiersprachen zu unterstützen. Insbesondere gilt dies für parallele Unterprogrammaufrufe und allgemeine Rekursion.

5. Dynamische Datenflußrechner

Der Token-Speicher müßte im Prinzip ein assoziativer Speicher sein, da ein inhaltsbezogener Zugriff verlangt ist. Da derartige Speichermedien sehr teuer sind, wird bei allen derzeitigen Datenflußrechnern ein pseudo-assoziativer Zugriff über Hash-Tabellen hergestellt. Die Hashfunktion wird auf den Tag und die Zieladresse des Token angewendet.

Das Problem der explodierenden Parallelität, auch Throttling Problem genannt, kann auf zwei Weisen gelöst werden:

- auf statische Weise durch Analyse der erzeugten Parallelität zur Compilezeit und durch entsprechende Maßnahmen zur Parallelitätsbegrenzung durch den Compiler, oder
- auf dynamische Weise durch Beobachtung der Auslastung der kritischen Ressourcen und durch automatische Parallelitätsbegrenzung zur Laufzeit der Maschine.

5.1 U-Interpreter

Eine Ausführung eines Operationsknotens wird als Aktivität bezeichnet.

Um Aktivitäten, die zu verschiedenen Iterationen oder Unterprogrammaktivierungen gehören, unterscheiden zu können, ohne den Code tatsächlich kopieren zu müssen, werden die Aktivitätsnamen um eine Iterationsnummer und eine Identifikation der Unterprogrammaktivierung erweitert. Jede Schleife und jedes Unterprogramm wird als separater Codeblock gespeichert.

Der U-Interpreter versieht jede Aktivität mit einem Aktivitätsnamen (oder Tag). Jedes Token transportiert einen Operanden einer Aktivität und trägt den Namen dieser Zielaktivität als Tag. Beim U-Interpreter besteht das Tag aus vier Feldern:

- Kontextfeld identifiziert eine Codeblockaktivierung, d.h. Kontextnummer zur Identifizierung des Unterprogrammaufrufs
- Codeblockname identifiziert einen Codeblock, der das Unterprogramm oder die Schleife enthält
- Befehlsindex identifiziert den Befehl innerhalb des Codeblocks
- Interationsnummer identifiziert die Schleifeniteration

5.2 I-Strukturen

Sind Speicherelemente zur Verwaltung (Speicherung und Zugriff) großer Datenstrukturen. Es handelt sich um einen nicht-strikten Array-Typ. Es gibt spezielle Zugriffsfunktionen.

5.3 k-begrenztes Schleifenschema

Durch das Prinzip wird die Anzahl gleichzeitig aktivierter Schleifeniterationen auf eine konstante Zahl k beschränkt.

5.4 Prinzip des expliziten Tokenspeichers

Hauptprobleme dynamischer Datenflußrechner sind effiziente Vergleichseinheiten. Man kann mit dem o.g. Prinzip den assoziativen Zugriff eliminieren. Dazu werden Schleifeniterationen und Funktionsaktivierungen jeweils ein eigener Aktivierungsrahmen zugeordnet. Bei jeder Aktivierung des Codeblocks wird ein eigener Aktivierungsrahmen eingerichtet. Die Speicherstellen in diesem Rahmen sind für die wartenden Tokens. Der Compiler legt für jeden Befehl eine Offset-Adresse relativ zu diesem Rahmen fest. Dadurch wird jedes Token, das zu einer Aktivierung dieses

Befehls gehört an der Offsetadresse in dem Aktivierungsrahmen seiner Iteration eingetragen. Assoziative Suche entfällt.