Die ReAl Computerarchitektur

ReAl = Ressourcen-Algebra

Ein einführender Überblick

Stand: 1.1 vom 11. 9. 2006

Prof. Dr. Wolfgang Matthes
Peukinger Weg 34
59423 Unna/Germany
http://www.realcomputerarchitecture.com

Mal was Neues ausprobieren . . .
Die derzeit gängigen Prozessoren beruhen auf Architekturprinzipien, die in den 70er und 80er Jahren entwickelt wurden. (Das betrifft sinngemäß auch die Betriebssysteme und Programmiersprachen.)

Seinerzeit war die Hardware knapp. Allen Entwurfsentscheidungen (welche architekturseitigen Vorstellungen werden verwirklicht, welche nicht?) mußte stets die Realisierbarkeit im Rahmen der jeweiligen Technologie zugrunde gelegt werden. Die heutigen Halbleitertechnologien ermöglichen es, mehrere herkömmliche Hochleistungsprozessoren auf einem einzigen Schaltkreis anzuordnen. Es ergibt sich die Frage, ob man diese Möglichkeiten nicht auch nutzen sollte, um etwas radikal Neues auszuprobieren . . .

Lohnt es sich, die Architektur des herkömmlichen Universalrechners weiter zu verbessern? – Kann sein, aber . . . In der Praxis hat bisher immer die Technologie die Leistung gebracht – architekturseitige Verbesserungen waren nur selten wirklich entscheidend. Die eigentlichen Wundertaten werden vom Silizium, von den Compilern und von den Algorithmen verrichtet . . . Architekturseitige Verbesserungen haben einen viel geringeren Effekt als die Weiterentwicklungen der Techologie:

- Die vielen Transistoren und GHz helfen über architekturseitige Unzulänglichkeiten hinweg.
- Da ohnehin weitgehend in höheren Sprachen programmiert wird, ist die Eleganz der Maschinenbefehlsformate und der architekturseitigen Wirkprinzipien im Grunde bedeutungslos.

Deshalb ein anderer Ansatz: Es ist genügend da (nicht wie bei armen Leuten . . .)[1)]:

- Hardware spielt keine Rolle,
- Speicherkapazität spielt keine Rolle,
- rechentechnische Voraussetzungen der Maschinenprogrammfertigung (z. B. mittels Compiler) spielen keine Rolle.

Unser Grundmodell:
Wenn wir irgend etwas tun wollen, so holen wir ein dazu geeignetes Stück Hardware aus einem Lager (wie einen Hammer, um einen Nagel einzuschlagen, einen Schraubenschlüssel, um eine Mutter festzuziehen usw.) und verwenden es für die auszuführende Informationswandlung. Wenn wir zwei Zahlen zueinander addieren wollen, holen wir einen Addierer, wenn wir zwei Werte miteinander vergleichen wollen, einen Vergleicher usw. Ein Stück Hardware, das seine Arbeit getan hat, wird in das

---

1): Mit anderen Worten: Klotzen statt Kleckern (H. Guderian).

Lager zurückgestellt. In Weiterführung der Analogie holen wir für unsere Arbeit soviele Werkzeuge, wie wir jeweils brauchen, z. B. 50 Hämmer, wenn 50 Nägel einzuschlagen, oder 50 Addierer, wenn 50 Zahlenpaare zueinander zu addieren sind.

Unsere Architekturdefinition umfaßt eine Menge von Ressourcen und eine Menge von Datenstrukturen. Ressourcen führen bestimmte Operationen über Daten bestimmter Strukturen aus.

Diese Abbildung konstituiert im Grunde eine algebraische Struktur. Deshalb wird die Architektur mit *ReAl = Re*ssourcen-*Al*gebra bezeichnet.

Das Modell einer Ressource ist eine Hardware, die bestimmte Informationswandlungen ausführt, also aus gegebenen Daten (an den Eingängen) neue Daten (an den Ausgängen) errechnet (Abb. 1).

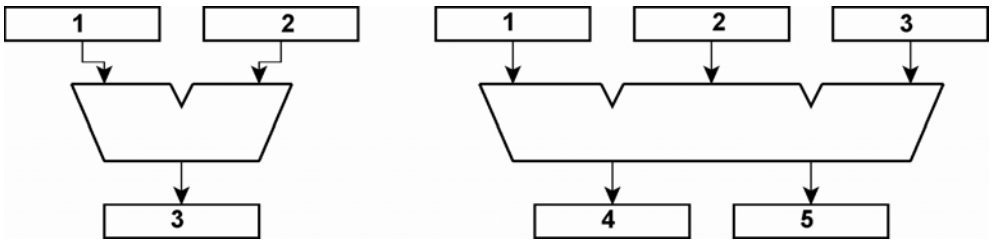

Abb. 1 Zwei Ressourcen im elementaren Modell

Allgemeine Annahmen und Entwicklungsziele:

1. Es gibt jederzeit genügend Ressourcen. Das ist zunächst eine theoretische Annahme (Hypothese vom (nahezu) unbeschränkten Ressourcenvorrat). Auf dieser Grundlage ist es möglich, Ressourcen in beliebiger Anzahl anzufordern (beispielsweise einige hundert Multiplikationswerke) und den inhärenten Parallelismus in vollem Umfang auszunutzen. Maschinenprogramme werden typischerweise so erzeugt (z. B. mittels Compiler), als ob beliebig viele Ressourcen zur Verfügung stünden. Die Anpassung an die Gegebenheiten der Praxis (jeder tatsächliche Ressourcenvorrat ist begrenzt) kann zur Compilierzeit oder zur Laufzeit erfolgen (Emulation, Virtualisierung).
2. Ob eine Ressource als Software oder als Hardware ausgeführt ist, spielt keine Rolle.
3. Das grundsätzliche Modell einer Ressource ist stets eine Hardware, also eine technische Einrichtung mit Ein- und Ausgängen (vgl. Abb. 1).
4. Ein Verarbeitungsvorgang (Programmablauf) besteht in der Benutzung von Ressourcen im Laufe der Zeit (Ressourcen werden bei Bedarf aus dem Ressourcenvorrat entnommen und bei Nichtgebrauch zurückgegeben).
5. In Hinsicht auf ein gegebenes Anwendungsproblem ist der Universalrechner im Grunde nur eine Notlösung. Um ein wirkliches Maximum an Verarbeitungsleistung zu erhalten, wäre eine Spezialhardware erforderlich, deren Maschinenzyklen ausschließlich dazu dienen, die gewünschten Ergebnisse zu berechnen. In einer solchen Maschine würden keine Taktzyklen und Speicherzugriffe dazu verwendet werden, Befehle zu holen, Zwischenwerte abzuspeichern und wieder zu laden, Funktionsaufrufe auszuführen usw. Im Grunde nehmen wir den Universalrechner nur deshalb, weil wir eine solche Maschine nicht ohne weiteres bauen können. Die ReAl-Architektur soll es ermöglichen, solche (fiktiven) Maschinen zur Laufzeit des Programms bedarfsweise auf- und abzubauen.
6. Die zur Steuerung der Verarbeitungsvorgänge vorgesehenen Anweisungsangaben (Operatoren) betreffen nur die grundlegenden Verfahrensschritte des Anforderns, Transportierens, Auslösens usw., nicht aber konkrete Maschinenoperationen (vollständige Maschinenunabhängigkeit).

7. Komplexere Ressourcen können rekursiv aus elementaren Ressourcen aufgebaut werden.
8. Wo die Ressourcen angesiedelt und wie sie aufgebaut sind, spielt keine Rolle. Unser Modell schließt u. a. auch die Möglichkeit ein, Ressourcen übers Internet anzufordern und auszunutzen (z.B. Spezialrechner).

Die Programmierphilosophie läuft letzten Endes darauf hinaus, eine Hardware zu entwerfen, die die betreffende Verarbeitungsaufgabe ausführen kann, und zwar zunächst als Gedankenexperiment, unabhängig von der tatsächlichen praktischen Durchführbarkeit.

Wir tun so, als ob sich für jede x-beliebige Informationswandlung eine Hardware bauen ließe und entscheiden dann fallweise, was tatsächlich als Hardware gebaut wird und was nicht.

Diese (fiktive) Hardware kann zur Laufzeit dynamisch auf-, um- und abgebaut werden. Die Maschinenbefehle der ReAl-Architektur betreffen keine bestimmten Operationen, sondern das Auf-, Um- und Abbauen von Ressourcen, die zugehörigen Datentransporte (Parameterversorgung) und das Aktivieren der betreffenden Ressourcen.

Um eine bestimmte Programmierabsicht auszuführen, werden jeweils geeignete Ressourcen aus der Menge aller Ressourcen (dem Ressourcenvorat) ausgewählt. Diese werden mit Parametern versorgt. Dann werden die Verarbeitungsvorgänge in den Ressourcen ausgelöst. Anschließend werden die Ergebnisse zugewiesen (Endergebnisse werden gespeichert oder ausgegebenen, Zwischenergebnisse an andere Ressourcen weitergeleitet). Weitere Schritte der Parameterversorgung, Auslösung und Zuweisung werden so oft durchlaufen, bis die jeweilige Verarbeitungsaufgabe ausgeführt ist. Schließlich werden die nicht mehr benötigten Ressourcen an den Ressourcenvorrat zurückgegeben. Diese Abläufe werden durch Programme gesteuert. Die Programmsteuerung beruht auf speziellen Maschinenbefehlen, den sog. Operatoren. Besondere Operatoren ermöglichen es, Verbindungen zwischen Ressourcen einzurichten (die Ressourcen miteinander zu verketten) und solche Verbindungen wieder zu trennen. Ist eine Verbindung (Verkettung) zwischen Ressourcen eingerichtet, laufen die Schritte der Parameterversorgung, des Auslösen der Verarbeitungsvorgänge und des Zuweisens der Ergebnisse innerhalb der verketteten Ressourcen selbsttätig ab. Durch Verketten von Ressourcen kann man – je nach Bedarf – fiktive Spezialprozessoren bauen, die dem Datenflußschema der jeweiligen Verarbeitungsaufgabe entsprechen.

Die Architekturprinzipien können implementiert werden:

1. mit herkömmlichen Universalrechnern (Emulation),
2. mit abgewandelten Universalrechnern (geänderter Befehlsdecoder, geänderter Registersatz, andere Mikroprogramme usw.),
3. mit programmierbaren Schaltkreisen (z. B. FPGAs),
4. mit spezieller, von Grund auf für diese Architektur ausgelegter Hardware.

Die Abb. 2 bis 5 zeigen typische Ressourcenanordnungen im Blockschaltbild.

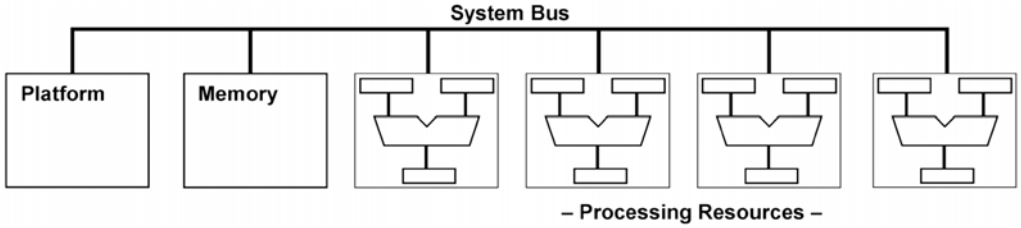

Abb. 2 Ressourcenanordung mit Universalbus

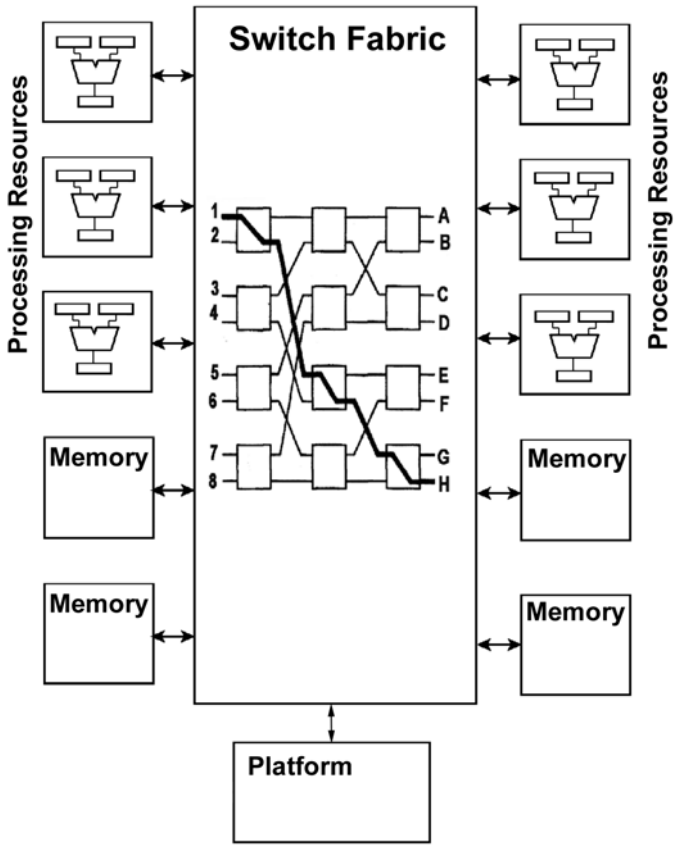

Abb. 3 Ressourcenanordung mit Schaltverteiler

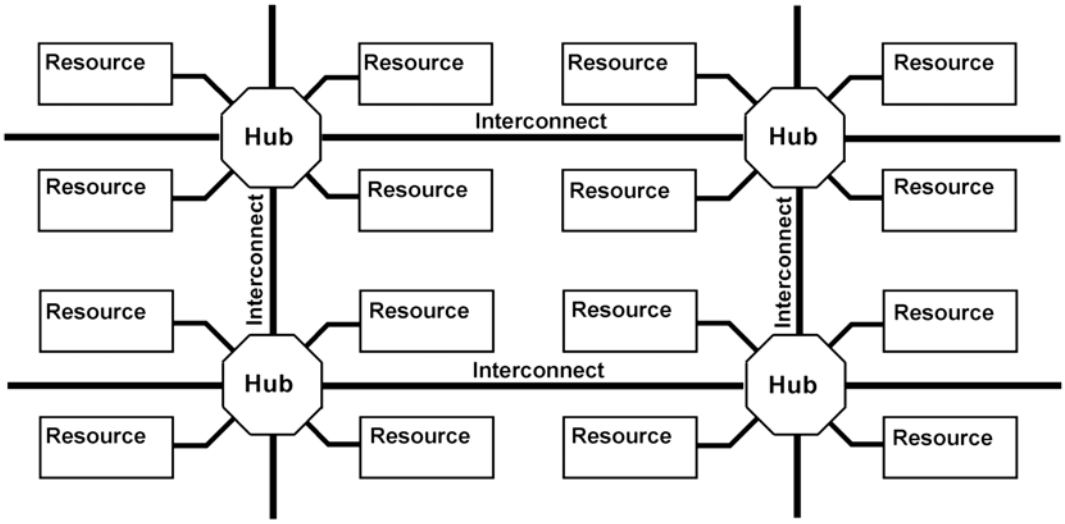

Abb. 4 Ressourcenanordung auf einem programmierbaren Schaltkreis

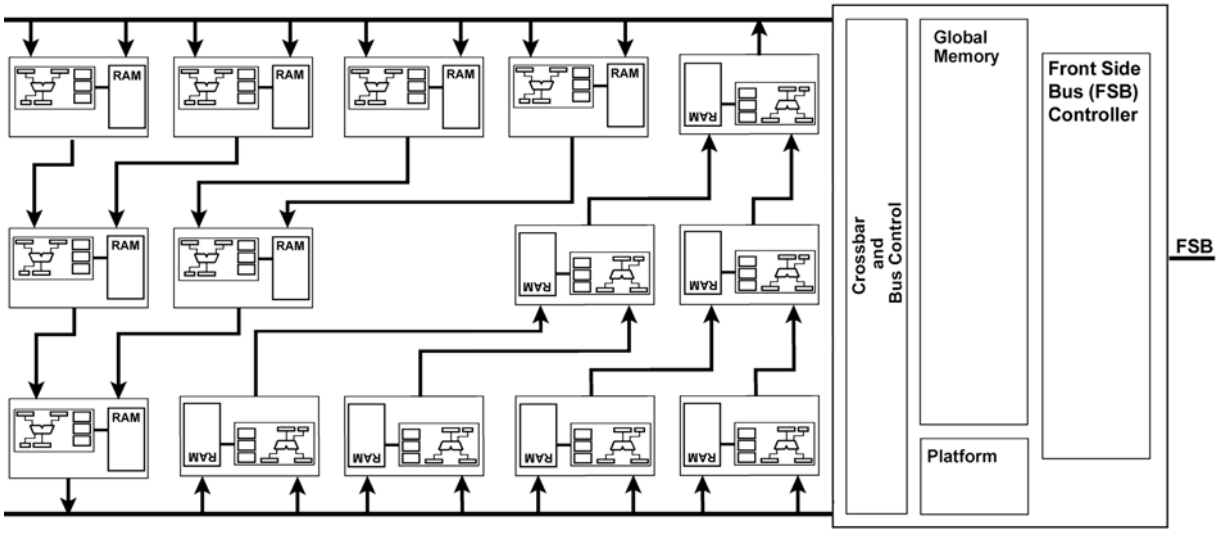

Abb. 5 Ressourcenanordung auf einem Schaltkreis. Im Beispiel sind jeweils sieben Ressourcen zu einem invertierten Binärbaum zusammengeschaltet

Durch Anwendung des ReAl-Programmiermodells kann der den Verarbeitungsabläufen innewohnende (inhärente) Parallelismus in dem Maße ausgenutzt werden, in dem tatsächlich Hardware verfügbar ist. Schaltmittel zur Konflikterkennung, Ablaufwiederholung usw. sind nicht erforderlich. Speicher und Verarbeitungsschaltungen können direkt miteinander verbunden werden (im Vergleich zu den Registersätzen der bekannten Hochleistungsprozessoren sind weniger Zugriffswege erforderlich, und die Adreßdecodierung ist einfacher). Verarbeitungsschaltungen können in Speicheranordnungen eingebettet werden (Ressourcenzellen, aktive Speicheranordnungen), so daß sich kürzeste Zugriffswege ergeben. Diese Vereinfachungen der Hardware bieten die Möglichkeit, die Taktfrequenz zu erhöhen, Pipeline-Stufe einzusparen (Verkürzung der Latenzzeiten der Operationsausführung) und auf einer gegebenen Schaltkreisfläche mehr bzw. leistungsfähigere Verarbeitungsschaltungen anzuordnen.

Da der inhärente Parallelismus unmittelbar aus der Programmierabsicht heraus (in statu nascendi) erkannt wird, ist es möglich, ggf. auch hunderte Verarbeitungswerke gleichzeitig einzusetzen, um den Ablauf des einzelnen Programms zu beschleunigen. Je nach den Kosten- und Leistungszielen und nach dem Stand der Technologie sind Hard- und Software gegeneinander austauschbar (z. B. ein Unterprogramm gegen eine spezielle Verarbeitungseinrichtung und umgekehrt). Entsprechende Programme sind somit invariant gegen die technologische Entwicklung; sie können jeden Fortschritt der Schaltungsintegration ausnutzen. Auf programmierbaren Schaltkreisen sind Systeme realisierbar, die eine im Grunde beliebige Kombinationen von Hard- und Software darstellen. Anwendungspraktisch wichtige Zusatzfunktionen, z. B. die Unterstützung des Fehlersuchens (Debugging), der Systemverwaltung, der Datenverschlüsselung usw., die herkömmlicherweise zusätzliche Programmabläufe (Geschwindigkeitsverlust) oder spezielle Hardware (Aufwand) erfordern, können in die Ressourcen organisch eingebaut werden.

Auf Grundlage der ReAl-Wirkprinzipien sollen verschiedene Arten künftiger Computersysteme entwickelt werden:

1. Neue Prozessoren für Embedded Systems. Solche Systeme können aus Universalrechnern und spezieller Hardware nach dem Baukastenprinzip konfiguriert werden.
2. Neue Prozessorschaltkreise als Alternative zu den derzeit vor allem von Intel und AMD angebotenen Hochleistungsprozessoren.

3. Echte Supercomputer mit einer Ressourcenausstattung, die dem jeweiligen Anwendungsproblem angemessen ist (es handelt sich weder um althergebrachte Vektorrechner noch um Prozessor-Farmen). Die Auslegung gemäß den ReAl-Wirkprinzipien ermöglicht:
   - das Aufrollen der innersten Schleifen,
   - die Ausnutzung des inhärenten Parallelismus, gleich auf welcher Ebene,
   - den Aufbau von Ressourcenkonfigurationen, die dem Datenflußschema des jeweiligen Anwendungsproblems entsprechen (fiktive Spezialmaschinen – hierdurch können auch datenabhängige Verarbeitungsabläufe unterstützt werden),
   - die Unterstützung der nicht parallelisierbaren Abläufe durch Spezialhardware,
   - die Minimierung des Communication Overhead.

Die ReAl-Wirkprinzipien ermöglichen es, die künftigen Möglichkeiten der Schaltungsintegration (z. B. einige hundert Millionen Transistoren auf einem Schaltkreis) in weitem Umfang auszunutzen. Herkömmliche Hochleistungsprozessoren haben rund 10...50 Millionen Transistoren. Auf einem Schaltkreis mit 200 Millionen Transistoren ließen sich z. B. vier Prozessoren zu jeweils ca. 50 Millionen Transistoren unterbringen. Das Leistungsvermögen dieser Anordnung kann aber nur dann in der Praxis wirksam werden, wenn wenigstens vier Programme gleichzeitig auszuführen sind; das einzelnen Programm kann man hiermit nicht beschleunigen. Zerlegt man einen einzelnen Prozessor in seine Funktionseinheiten, so entsprechen die Verarbeitungseinrichtungen 4...8 universellen Rechenwerken. Wenn Caches, Steuerschaltungen usw. – ihrem Umfang nach – beibehalten werden (gleicher Aufwand, nur abgewandelte Struktur), so könnte ein Schaltkreis mit 200 Millionen Transistoren 16...64 universelle Rechenwerke enthalten, die als Ressourcen verwaltet werden und somit jedem einzelnen Programm zugute kommen können. Anwendungsbeispiel: Ein Prozessor für Spielkonsolen, der – abhängig vom Zustand des aktuellen Spiels – als Graphikmaschine, Physikmaschine, KI-Maschine, Datenbasismaschine usw. eingerichtet wird.

Das ReAl-Programmiermodell läßt sich auch in Compilern anwenden. Ein typischer Entwicklungsgang: Formulieren der Programmierabsicht (in einer beliebigen Programmiersprache) => Wandlung in den Code einer fiktiven ReAl-Maschine => Umsetzung in den Maschinencode der Zielarchitektur => Programmausführung. Es ist üblich, den Quellcode zunächst in einen fiktiven Maschinencode umzusetzen. Solche fiktiven (virtuellen) Maschinen sind zumeist als Stackmaschinen ausgelegt. Stackmaschinen arbeiten aber inhärent sequentiell. Demgegenüber haben gemäß den ReAl-Architekturprinzipen ausgelegte fiktive Maschinen vor allem folgende Vorteile:

1. Der inhärente Parallelismus im Programmablauf kann praktisch vollständig erkannt werden.
2. Es ist möglich, Gelegenheiten zur Ausnutzung von SIMD-Vorkehrungen und VLIW-Befehlen in allgemeinen Programmabläufen zu erkennen.
3. Verringerung des Anteils der Organisationsabläufe (Overhead), z. B. beim Aufrufen von Funktionen und beim Transportieren von Parametern.

Sämtliche Programme, gleich in welcher Sprache formuliert, sind letzten Endes Steueranweisungen für Informationswandlungen, Transporte und Zustandsübergänge in Register-Transfer-Strukturen. Eine hinreichend mit Datentypen, Operationen usw. ausgestattete Ressourcenkonfiguration eignet sich somit als universelles Compiler-Ziel. Der Maschinencode einer solchen Konfiguration ist praktisch eine universelle Metasprache, in der alle Ausdrücke der verschiedenen Programmiersprachen wiedergegeben werden können. Ein typischer Entwicklungsgang: Programm in Programmiersprache A => Byte- oder Maschinencode der ReAl-Architektur => Programm in Programmiersprache B.

Fiktive (virtuelle) Maschinen auf Grundlage der ReAl-Architektur können als Gegenstück zur bekannten Java Virtual Machine (JVM) angesehen werden (Tabelle 1). JVM ist – im Verbund mit der Programmiersprache Java – vor allem für kleinere Programme (Applets) vorgesehen. Hierfür ist die Kompaktheit des Codes von besonderer Bedeutung. In einem kompakten Maschinencode läßt sich aber die Parallelverarbeitung nicht beschreiben. Die ReAl-Architektur betrifft hingegen die oberen

Leistungsbereiche. Hier kommt es vor allem darauf an, den inhärenten Parallelismus so gut wie möglich auszunutzen. Mit den Operatoren der ReAl-Architektur kann der einem Programmablauf inhärente Parallelismus in vollem Umfange dargestellt werden. Zudem ist es möglich, Programmabläufe in Datenflußschemata und fiktive (virtuelle) Spezialhardware umzusetzen.

Java, JVM	**ReAl**
• Kompaktheit des Codes (Bytecode), • vor allem kleine Programmstücke (Applets), • Lauffähigkeit auf kleinen Maschinen, • Laden der Programme übers Internet, • JVM ist herkömmliche Stackmaschine, also inhärent sequentiell, • JVM-Bytecode beschreibt eine Operation zu einer Zeit; der inhärente Parallelismus müßte ggf. zur Laufzeit eigens erkannt werden (Hardware – wie im Pentium –, Just-in-Time-Compiler o. dergl.)	• Maximale Ausnutzung der Hardware, • vor allem umfangreiche und verarbeitungsintensive Programme (Graphik, Numerik, Datenbanken, neuronale Netze, KI), • Speicherkapazität spielt keine Rolle, Code-Umfang spielt keine Rolle, Hardware ist genügend da. Wir brauchen zunächst einmal mehr – das aber wird sich bezahlt machen ... (Optimieren können wir später immer noch.) • Lauffähigkeit u. a. auf Maschinen, die die künftige Schaltungsintegration zu bauen ermöglicht (dutzende...hunderte Rechenwerke auf einem Schaltkreis), • ReAl-Code beschreibt Programmabläufe unter voller Ausnutzung des inhärenten Parallelismus bis hin zum Aufbau fiktiver Spezialprozessoren, die dem Datenflußschema der Verarbeitungsaufgabe entsprechen. Inhärenter Parallelismus wird in statu nascendi (= aus dem ursprünglichen Programmtext heraus) erkannt und nicht erst zur Laufzeit

Tabelle 1 Java Virtual Machine (JVM) vs. Real

Ein auf der ReAl-Architektur beruhendes Programm kann die Programmierabsicht in allen wesentlichen Einzelheiten beschreiben; wenn es sein muß, bis hin zur einzelnen Booleschen Gleichung. Deshalb ist zu erwarten, daß sich solche Programme ohne weiteres in Maschinencode für künftige Systeme umsetzen lassen (Zukunftssicherheit).

Die Architekturprinzipen können ausgenutzt werden, um Anwendungen zu unterstützen, an die hohe Sicherheitsanforderungen gestellt werden. Es seien zunächst zwei Möglichkeiten genannt:

1. Es werden Ressourcen geschaffen, in die entsprechende Kontrollvorkehrungen (Überwachung von Wertebereichen, Signalisierung von Ausnahmebedingungen, Verschlüsselung usw.) von Grund auf eingebaut sind.
2. Anstelle eines herkömmlichen Programms wird zunächst eine fiktive Schaltung erzeugt; die Programmierabsicht wird in eine fiktive Schaltungsstruktur umgesetzt, deren Informationswandlungen bis auf die einzelnen Booleschen Gleichungen aufgelöst werden können. Hierdurch werden formale Korrektheitsbeweise vereinfacht bzw. überhaupt erst durchführbar (Anwendung der Graphentheorie, Automatentheorie, Schaltalgebra usw.). Der Vorteil wird auch dann wirksam, wenn die Architektur auf herkömmlichen Prozessoren emuliert wird. (Ein richtig geschriebener Emulator kann nie abstürzen, ganz gleich, welche Fehler im zu interpretierenden System vorhanden sind. Somit hat derartige Software die gleiche Funktionssicherheit wie eine echte Hardware-Implementierung.)

Die ReAl-Architektur ermöglicht es, komplexe Entwürfe unabhängig von ihrer Realisierung zu beschreiben und die Implementierung mit universellen Schaltungen, Spezialschaltungen oder Software je nach Zweckmäßigkeit festzulegen. Aus den Ressourcen (Rechenschaltungen, Adressierungsschaltungen usw.) kann man je nach Bedarf Universalrechner oder Spezialschaltungen aufbauen und die Konfiguration während der laufenden Arbeit ändern.

Kennzeichnend ist die Auflösung der Prozessorstrukturen in die einzelnen Funktionseinheiten und der fließende Übergang zwischen Hard- und Software. Die festen – proprietären – Prozessorarchitekturen, Betriebssysteme und Anwendungsprogramme können durch Ressourcen an sich beliebiger Herkunft ersetzt werden (aufgelöste Systemarchitektur). Sowohl System- als auch Anwendungsfunktionen werden von Ressourcen erbracht, die wahlweise als Hardware oder als Software ausgeführt werden können. Wesentlich hierfür ist, daß die Operatoren nur das Aufrufen, Aktivieren usw. der Ressourcen beschreiben, während die eigentlichen funktionellen Wirkungen im Innern der jeweiligen Ressource erbracht werden.

Anhängige Patentanmeldungen:

- DE 10 2005 021 749.4 “Verfahren und Vorrichtung zur programmgesteuerten Informationsverarbeitung",
- US 11/430,824 “Method for Information Processing“.