Codesigned Virtual Machines

Seminar Virtualisierung

Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur

Philipp Kirchhofer
Universität Karlsruhe

***Zusammenfassung*—In dieser Ausarbeitung, entstanden im Rahmen des Seminars „Virtualisierung“, werden die möglichen Gestaltungsmöglichkeiten durch den Einsatz von Codesigned Virtual Machines (CVMs) erläutert. Als Einführung werden die Konzepte des Codesigns und der Codesigned Virtual Machine vorgestellt, anschließend wird auf die grundlegende Struktur von CVMs eingegangen. In späteren Abschnitten werden Probleme und Lösungen für den Praxiseinsatz diskutiert. Auf den Praxiseinsatz von CVMs wird durch die detaillierte Darstellung von zwei kommerziellen Systemen (Transmeta Crusoe/Efficeon, IBM System i) besonders eingegangen. Ergänzend steht eine umfangreiche Literaturliste als Vertiefungsgrundlage zur Verfügung.**

I. Einführung

Der Entwurfsprozess für eine Rechnerarchitektur ist normalerweise streng strukturiert, es wird als Erstes die Hardware und die dazugehörige Befehlssatzarchitektur (Instruction Set Architecture (ISA)) entworfen und anschließend die Software angepasst bzw. neu entwickelt. Ein Problem ist, dass es mittlerweile viele Betriebssysteme und Anwendungsprogramme für alte ISA gibt, im Consumer Bereich vor allem die IA-32 ISA. Diese Architektur ist bedingt durch ihr Alter recht eingeschränkt (Nur 8 frei verfügbare Register, CISC artiger Aufbau). Moderne superskalare Consumer Prozessoren (Pentium 4, Athlon 64) müssen deshalb den Befehlscode in kleinere Einheiten (Microoperationen) zerlegen, um eine akzeptable Performance erreichen zu können (Out-of-Order Ausführung). Das Aufteilen in Microoperationen und das Verteilen (Scheduling) derselben auf die Funktionseinheiten des Prozessors (ALU, FPU) benötigt viele Transistoren, damit auch elektrische Leistung, und bedingt zu einem großen Teil die Komplexität des Prozessordesigns. Ein Ersatz der eingesetzten ISA durch eine neue effizientere und performantere, allerdings nicht rückwärtskompatible, ISA ist vor allem aufgrund der fehlenden Binärkompatibilität mit bestehenden Betriebs- und Anwendungssystemen selten erfolgreich (siehe auch Intel Itanium und Intel i860).

Eine andere Herangehensweise ist der gemeinsame Entwurf von Hardware und Software in einem Schritt. Dieses Codesign genannte Konzept ermöglicht große Flexibilität im Aufbau der Gesamtarchitektur und erlaubt es, in Verbindung mit einer Virtuellen Maschine als Softwarekomponente, eine „alte“ (Legacy) Befehlssatzarchitektur weiterhin zu unterstützen. Weiterhin kann um maximale Performance und Effizienz zu erreichen die Host ISA weitgehend unabhängig von den Einschränkungen der Gast ISA aufgebaut werden.

A. Gründe für Codesigned Virtual Machines (CVMs)

CVMs werden eingesetzt, um verbesserte Performance durch den Einsatz von neuen Techniken zu erreichen. Weiterhin kann auch eine höhere Energieeffizienz im Vergleich zu herkömmlichen Prozessoren erzielt werden, da das eigentliche Hardwaredesign durch den Wegfall der Bearbeitung von Mikrooperationen einfacher aufgebaut ist und besser optimiert werden kann (Keine CISC-RISC Umwandlung in Hardware). Kompatibilität ist eine notwendige Bedingung, allerdings grundsätzlich keine Motivation für das Verwenden von CVMs.

CVMs ähneln strukturell den konventionellen superskalaren Prozessordesigns: Beide Arten wandeln Instruktionen von einer Gast ISA in eine in Hardware implementierte Host ISA um. Der Hauptunterschied ist der Ort der Umwandlung: Bei ersteren findet die Umwandlung in Software statt. Dies ermöglicht weitergehende Optimierungen für die Instruktionsumwandlung, da der Kontext einer Instruktion mit in die Umwandlung einfließen kann. Bei letzteren wird üblicherweise jede Gast-Instruktion unabhängig von allen weiteren Gast-Instruktionen behandelt, es kann also keine Kontextsensitive Optimierung durchgeführt werden.

II. Technische Implementierung

In diesem Kapitel werden wichtige Techniken und Vorgehensweisen für die Umsetzung von CVMs in Hard- und Software vorgestellt. Weiterhin wird auf einige der häufigsten Probleme eingegangen.

A. Virtual Machine Monitor (VMM)

Ein zentraler Bestandteil der CVM ist der VMM. Er führt die Umwandlung der Instruktionen der Gast ISA durch, sorgt für Cache-Kohärenz (siehe Kapitel „Code-Caching“), die korrekte Behandlung von Traps und Unterbrechungen (siehe Kapitel „Checkpointing“) und läuft selber als einziges Programm direkt auf der Host ISA.

B. Register Abbildung

Die Abbildung von Registern der Gast ISA auf Register der Host ISA ist eine essentielle Funktionalität der VM. Üblicherweise stellt die Host ISA mehr Register als für die Umsetzung benötigt bereit. Die zusätzlichen Register können zur Optimierung der Übersetzung, für VM Verwaltungszwecke und als Schattenregister verwendet werden.

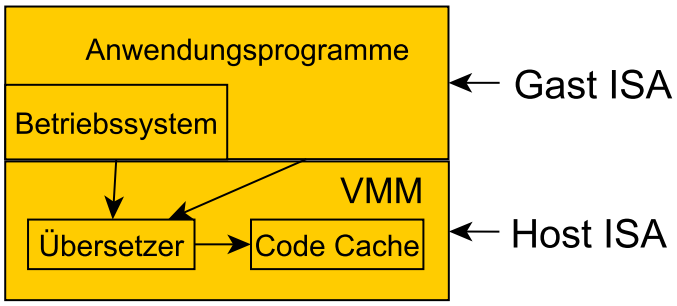

Abbildung 1. Virtual Machine Monitor (nach [1])

C. Speicher Abbildung

Der reale Systemspeicher wird bei der Speicher Abbildung in zwei disjunkte Teile aufgeteilt:

1) Reservierter Speicher für VMM (Nicht verfügbar für Betriebssystem):
   a) Übersetzungscache
   b) Verwaltungsinformationen
   c) Code und Daten der VMM Software
2) Konventioneller Speicher für Betriebssystem und Anwendungsprogramme:
   a) Betriebssystem
   b) Anwendungen

Das Betriebssystem kann nicht auf den für den VMM reservierten Speicher zugreifen. Sobald das Betriebssystem startet, wird die Abfrage des verfügbaren Speichers abgefangen und ein um die Größe des reservierten Speichers verminderter Wert zurückgeliefert. Weiterhin führt ein Zugriffsversuch von Programmen auf den reservierten Speicher zu einem Speicherzugriffsfehler, analog zu einem Zugriff auf nicht existenten Speicher. Die Existenz des VMM ist damit sowohl für das Betriebssystem als auch für die Anwendungsprogramme vollständig transparent.

D. Selbstmodifizierender Code

Jeglicher Anwendungs- bzw. Betriebssystemcode wird als schreibgeschützt betrachtet. Die Translation Lookaside Buffer (TLB)s enthalten dazu ein spezielles Schreibschutz-Bit, das für alle Einträge, die auf Speicher aus dem konventionellen Code-Speicherbereich zeigen, gesetzt ist. Bei einem Schreibzugriff auf eine Seite mit gesetztem Schreibschutz-Bit kann der VMM den Übersetzungscache für die jeweilig betroffene Seite leeren und den Übersetzungsvorgang neu anstoßen.

E. Code Caching

Da Programme oft zeitlich und örtlich lokal ablaufen, bietet es sich an die übersetzten Instruktionen in einem Code Cache für den schnellen Zugriff zwischenzuspeichern. Der Zugriff auf diesen Cache ist für die Performance und Effizienz der CVM von größter Wichtigkeit. Im Gegensatz zu Caches mit festen Blockgrößen, wie z.B. in herkömmlichen Prozessoren (L1-, L2-Cache) eingesetzt, ist die Blockgröße des Code Cache variabel. So können lange, sequentielle Programmabschnitte als großer Block am Stück abgespeichert werden. Programmabschnitten mit vielen Sprüngen können dagegen in kleine, unabhängige Blöcke umgesetzt werden.

F. Checkpointing

Bestimmte Architekturen, wie z.B. IA-32, erfordern präzise Ausnahmesemantiken: Wenn eine Instruktion eine Ausnahme auslöst müssen alle davorliegenden Instruktionen abgeschlossen und alle darauffolgenden bereits angefangenen Instruktionen verworfen werden bevor die Ausnahme weiterverabeitet werden kann. Da bei CVMs Instruktionen häufig blockweise ausgeführt werden ergeben sich Probleme, wenn innerhalb eines Blocks eine Ausnahme ausgelöst wird: Eine Ausnahme kann erst nach vollständiger Ausführung eines Blockes verarbeitet werden. Dabei wird allerdings die oben spezifizierte präzise Ausnahmesemantik verletzt, da Instruktionen, die nach der Ausnahmenauslösenden Instruktion vorkommen, bereits parallel ausgeführt wurden.

Eine performante Lösung für dieses Problem ist das sogenannte „Checkpointing“. Dabei werden alle relevanten, d.h. den Zustand der CPU bestimmenden Register in Schattenregistern gespiegelt. Es existieren also jeweils zwei Kopien eines Registers: Eine Arbeitskopie und eine Schattenkopie. Während der Ausführung eines Blocks werden nur die Arbeitskopien verändert. Nach erfolgreicher Ausführung des Blocks kopiert eine „Commit“ Operation alle Arbeitskopien der Register in die jeweils passenden Schattenregister. Damit wird das Ergebnis des Blocks festgeschrieben. Beim Auftreten einer Ausnahme während der Ausführung eines Blocks werden durch eine „Rollback“ Operation, die alle Arbeitskopien mit den gespeicherten Schattenregisterwerten überschreibt, alle Änderungen zum vorherigen Block rückgängig gemacht. Anschließend kann der VMM Fehlerbehandlungsmechanismen wie z.B. Instruktionsweise Ausführung einleiten.

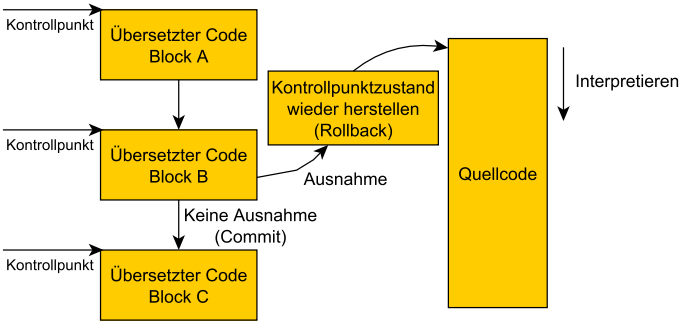

Abbildung 2. Hardware unterstütztes Checkpointing (nach [1])

Eine Variante des Checkpointings kann auch auf Speicheroperationen angewendet werden: Zu speichernde Daten werden in einem „Gated Store Buffer“ zwischengespeichert. Bei erfolgreicher Ausführung eines Blocks werden die Daten freigegeben und in den Hauptspeicher geschrieben. Nach Auftreten einer Ausnahme kann der Puffer einfach geleert werden.

Mit beiden Varianten kann damit sowohl CPU- als auch Speicherkonsistenz bei blockweiser Ausführung sichergestellt

werden.

Anmerkung: Der Ansatz des Checkpointing ähnelt in seinen Grundzügen dem Transaktionsorientierten Ansatz zur Konsistenzsicherung bei Datenbankensystemen.

III. Anwendung in der Praxis

A. Transmeta

1) Entstehungsgeschichte: Transmeta wurde 1995 als Stealth Startup in Santa Clara, Kalifornien gegründet und entwickelte die Prozessorreihen Crusoe und dessen Nachfolger Efficeon. CEO war David Ditzel, der 15 Jahre früher mit dem zusammen mit David Patterson verfassten Artikel „The Case for the Reduced Instruction Set Computer“ [2] einen Boom an RISC Designs auslöste. Im Gegensatz zu damals entstand bei Transmeta ein Prozessor nach dem CVM Ansatz, der auf einer internen VLIW Architektur basierte.

Die Zielsetzung beim Design der Prozessoren deckte sich überwiegend mit den Zielen, die für den Einsatz von Virtuellen Maschinen sprechen. Besonderen Wert wurde allerdings auf eine energieeffiziente Arbeitsweise gelegt, da Mobilprozessoren im Consumerumfeld zur damaligen Zeit sehr ineffizient waren und Transmeta in diesem Marktsegment eine Marktlücke sah.

2) Code Morphing Software: Die Code Morphing Software (CMS) ist Transmetas Bezeichnung für den bei den Crusoe und Efficeon CPUs eingesetzten VMM. Die Software interpretiert Instruktionen einer Gast ISA und kann diese auch in Instruktionen für den VLIW Prozessorkern übersetzen. Die Gast ISA kann prinzipiell beliebig gewählt werden, Transmeta entschied sich allerdings, nur die weit verbreitete IA-32 Architektur zu unterstützen. Eine einfache Übersicht über die Funktionsweise der CMS ist in Abbildung 3 dargestellt.

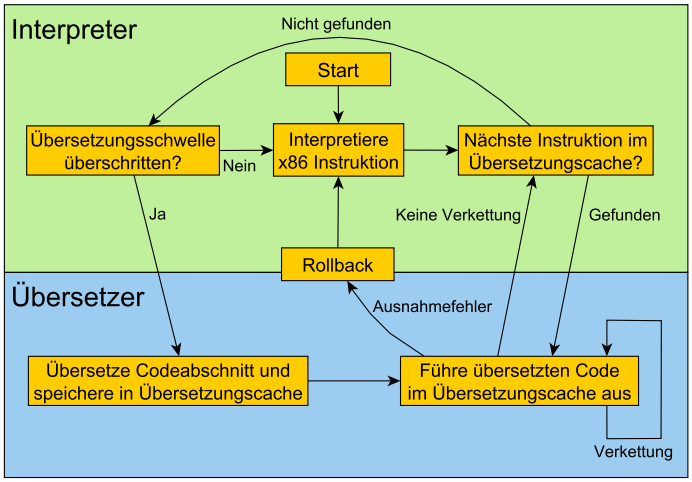

Abbildung 3. Transmeta CMS Schema (nach [3])

Jede x86 Instruktion wird in ein oder mehrere Atome zerlegt. Diese Atome werden anschließend in 128 Bit großen Molekülen zusammengefasst, dabei können in einem Molekül auch Atome aus verschiedenen Ursprungs-x86-Instruktionen verwendet werden. Alle Atome eines Moleküls werden vom Prozessor parallel ausgeführt, die Atome eines Moleküls werden direkt auf die entsprechenden Funktionseinheiten des Prozessors verteilt, dadurch kann die Dekodierungs- und Verteilungslogik im Prozessor vereinfacht ausgeführt werden. Die Moleküle werden wiederum sequentiell ausgeführt (In-Order), deshalb ist es besonders wichtig, dass in einem Molekül möglichst viele Atom-Plätze belegt sind. Durch die Verwendung der In-Order Ausführung kann auf eine komplexe und aufwendige Out-Of-Order Hardware verzichtet werden.

Eine Beispielkonfiguration ist in Abbildung 4 zu sehen.

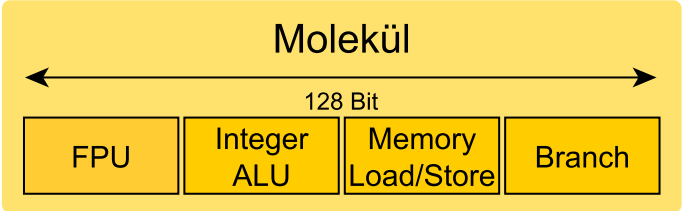

Abbildung 4. Aufbau eines Transmeta Crusoe Moleküls (nach [4])

Die CMS kann bekannte Hardware Bugs umgehen und erlaubt Performance Verbesserungen ohne Änderung der Hardware. Weiterhin reagiert sie dynamisch auf die auszuführenden Programme: Am Anfang wird jede Instruktion interpretiert, dies führt zu geringem Overhead, allerdings auch zu langsamer Ausführung. Sobald ein Programmteil häufig genug ausgeführt wird wird dieser Teil in native Instruktionen (Moleküle, s.o.) übersetzt und im Übersetzungscache gespeichert (vgl. Konzept „JIT Compiler“). Die Übersetzung dauert einige Zeit, weitere Ausführungen des Programmteils werden allerdings deutlich schneller durchgeführt. Die Behandlung eines Programmteils kann der jeweiligen Situation angemessen dynamisch im Laufe der Zeit zwischen Interpretation und Übersetzung wechseln.

Zur weiteren Optimierung können die übersetzten Programmteile verkettet werden: Sprünge zwischen übersetzten Programmteilen werden so angepasst, dass die Sprungziele direkt auf die jeweils passende Stelle im Code Cache verweisen. Häufig verwendete Programmteile können dadurch vollständig im Übersetzungscache ausgeführt werden, der Overhead für Interpretation und Übersetzung wird in solchen Fällen vollständig vermieden.

Um hohe Performance erreichen zu können optimiert die CMS aggressiv. Es werden folgende (nicht zwangsläufig zutreffende) Annahmen getroffen:

- Operationen lösen keine Ausnahmen aus
- Speicher Zugriffe sind nicht aliased
- Kein selbstmodifizierender Code

Durch diese Annahmen kann der eingehende Code neu angeordnet (Code Reordering/Scheduling) werden. Nach jedem ausgeführten Molekül müssen die Annahmen auf Korrektheit geprüft werden, um die korrekte Funktion des neu angeordneten Codes garantieren zu können. Treffen die Annahmen zu, kann der aktuelle Zustand als neuer Zustand verwendet und die Inhalte der Schatternregister verworfen werden (Commit), andernfalls werden die Änderungen durch ein Kopieren

der Schattenregister auf den Registersatz rückgängig gemacht werden (Rollback). In diesem Fall wird anschließend eine instruktionsweise Interpretation des x86-Codes angestoßen.

Die Anzahl der Verletzungen von Annahmen in einem Codebereich wird laufend überwacht, bei Überschreiten eines Schwellwertes kann der übersetzte Code mit weniger aggressiven Annahmen neu übersetzt werden. Weiterhin können auch einzelne Instruktionen, die viele Ausnahmen verursachen, isoliert übersetzt werden, damit der restliche Code weiter aggressiv optimiert werden kann.

B. MegaProto/E Cluster

Durch den Einsatz von CVMs ergeben sich neue Möglichkeiten für den Bau von energieeffizienten Prozessoren: Japanische Wissenschafter zeigten dies anschaulich mit dem Bau eines Cluster-Systems [5], bestehend aus Transmeta Crusoe/Efficeon Prozessoren, mit dem hohe Performance durch Nutzung von Niedrigenergieprozessoren bei hoher Packungsdichte erreicht wurde. Durch Entwicklung eines speziellen Mainboards konnten 16 Transmeta Efficeon Prozessoren in einem 19 Zoll Standard-Rack auf einer Höheneinheit untergebracht werden (Cluster Einheit, siehe Abbildung 5). Aufgrund der geringen Leistungsaufnahme konnten die Prozessoren passiv gekühlt werden, dabei lag die höchste erreichte Temperatur im laufenden Betrieb immer unter 40 °C. Die Performance einer Cluster Einheit überschritt bei allen Benchmarks (High Performance Linpack (HPL)/NAS Parallel Benchmarks (NPB)) die Performance eines Dual-Xeon Systems (siehe auch Tabelle I) um das 1,3 bis 3,69 fache. Weitere Versuche ergaben nach Optimierung der Netzwerkleistung zwischen mehreren Cluster Einheiten eine maximal erzielbare Leistung von 1 TFlops mit 10KW Energieverbrauch in einem 19 Zoll Standard-Rack.

Abbildung 5. MegaProto/E Cluster Einheit (aus [5])

C. IBM System i

Ein anderes Beispiel für eine Anwendung des Codesign-Ansatzes und der Verwendung von CVMs ist die IBM System i Architektur (früher bekannt als AS/400 oder iSeries).

	TDP	Peak Perf.	Peak Perf. / Power
Crusoe TM5800	7,5 W	930 MFLops	124 MFlops / W
Efficeon TM8820	3 W	2000 MFlops	666.7 MFlops / W
Intel Dual-Xeon*	170 W	12200 MFlops	71,76 MFlops / W

*Referenz CPU

Tabelle I
MegaProto Cluster Prozessoren

Bei der Entwicklung des Systems entwarf IBM die gesamte Hardware- als auch Software-Architektur und das dazugehörige Betriebssystem von Grunde auf neu. Im Gegensatz zu den Transmetaprodukten wählte IBM keine bereits bestehende ISA als Quell-ISA aus, sondern entwickelte eine komplett neue, hoch abstrahierte ISA, um Architektur-Unabhängigkeit und ein einfaches Design der darauf aufbauenden Software zu ermöglichen. Für den Entwurf der Prozessoren griff IBM auf die Hardwarebeschreibungssprache VHDL zurück. (Siehe auch [6])

Die neu entworfene ISA wird als technology-independant machine interface (MI) bezeichnet und bildet zusammen mit einer Standardbibliothek (Licensed Internal Code (LIC)), die sich mit den architekturabhängigen Aspekten der Ressourcenverwaltung beschäftigt, eine Binärschnittstelle (ABI), auf der die architekturunabhängige Software aufsetzt. Die ISA war für die damalige Zeit (1986) mit 305 spezifizierten Instruktionen sehr umfangreich, im Vergleich dazu besaß der damals aktuelle Intel Mikroprozessor 80386 dagegen nur knapp 140 Instruktionen.

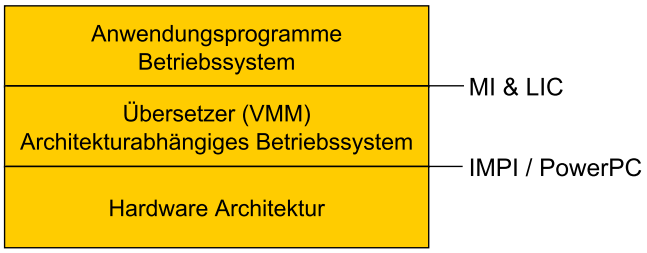

Abbildung 6. IBM System i Architektur (nach [1])

Ursprünglich basierte die zugrunde liegende Hardware Architektur auf einer proprietären CISC ISA namens „Internal Microprogrammed Interface“ (IMPI). Im Laufe der Zeit wurde diese durch eine PowerPC basierte ISA abgelöst. Aufgrund der Konstruktionsweise konnte dies ohne Austausch und Neukompilation der Anwendungssoftware und der Architekturunabhängigen Betriebssystemteile geschehen.

Das MI basiert auf einem objektorientierten Ansatz: Jeglicher Speicher (Haupt- und Hintergrundspeicher) ist in Objekte unterteilt. Im Gegensatz zu herkömmlichen Architekturen besteht nicht die Möglichkeit eines wortweisen Zugriffs auf den Speicher. Die Objekte sind voneinander unabhängig und können nur durch Zeigerreferenzierung verwendet werden. Eine Reihe von Objekttypen ist bereits in der MI Spezifikation vorgegeben, so gibt es z.B. spezielle Objekte für

Prozessmanagement wie Queues und Prozesskontrollblöcke und Objekte für Ein-/Ausgabe Verwaltung wie Controller oder Netzwerk Beschreibungen. Manche Objekttypen werden durch zusätzliche Instruktionen unterstützt.

Weiterhin ist das Speichersystem als einstufiger Speicher realisiert: Arbeitsspeicher und Festplattenspeicher werden im virtuellen Adressraum zusammengefasst. Zwischen den beiden Speichertypen gibt es programmatisch keine Unterscheidung, der Festplattenspeicher kann transparent als Arbeitsspeichererweiterung und zur Persistenzsicherung verwendet werden. Bei der Erstellung von Objekten kann die Art der Persistenz angegeben werden: Temporäre Objekte werden nur bis zur Systemabschaltung vorgehalten, Permanente Objekte stehen dagegen auch nach dem nächsten Systemstart zur Verfügung.

Bei der Konstruktion des MI wurde auf Zukunftssicherheit geachtet: Der Adressraum der System i Architektur ist auf 128 Bit Größe angelegt, d.h. alle Pointerobjekte sind 128 Bit lang. Diese Größe muss nicht unbedingt von den benutzten Prozessoren unterstützt werden: So konnten die ursprünglich verwendeten IMPI CPUs nur 48 Bit ansprechen, die derzeit verwendeten PowerPC CPUs können 80 Bit ansprechen. Durch die Dimensionierung kann allerdings bei Bedarf ohne Probleme ein Prozessor mit einem 128 Bit breiten Adressbus eingesetzt werden.

Anwendungsprogramme und das Betriebssystem werden allgemein als MI Code ausgeliefert. Bei Bedarf (Performance) kann die Übersetzungsoftware den MI Code in einen maschinenabhängigen Code bringen. Dabei wird das Programmobjekt um einen zusätzlichen Speicher für den übersetzten Code erweitert. Beim Umstieg auf eine neue Architektur ist das Programm weiterhin funktionsfähig, der neue maschinenabhängige Code wird durch Übersetzung aus dem weiterhin gespeicherten MI Code generiert.

IV. Ausblick

Im Consumer Bereich haben die Transmeta Crusoe/Efficeon CPUs gezeigt, dass der Einsatz einer CVMs zu einer erfolgreichen Kombination von akzeptabler Performance und niedriger Leistungsaufnahme führen kann. Finanziell war die Entwicklung für Transmeta allerdings nicht erfolgreich. Bis zum Ende der Prozessorproduktion Anfang 2005 schrieb Transmeta in jedem Quartal rote Zahlen. Transmeta konnte sich insbesondere gegen Intel bei der Belieferung von Notebookherstellern nie durchsetzen. Nach Aufgabe der Prozessorproduktion und bald darauffolgend auch der Prozessorentwicklung konzentrierte sich Transmeta auf die Lizensierung der Code Morphing Software und weiterer Technologien zur Effizienzsteigerung von integrierten Schaltungen (LongRun/LongRun2).

Im Midrange-Server Bereich ist die IBM System i Serie dagegen weiterhin erfolgreich im Einsatz. Durch die Möglichkeit des Einsatzes neuerer PowerPC Prozessoren kann davon ausgegangen werden, dass die Serie auch in den nächsten Jahren weiterentwickelt wird.

Weitergehende Forschungen an der Universität Wisconsin-Madison [7] zeigen, dass weiterhin Potenzial bei der Entwicklung von CVMs besteht. Weitere Produkte, besonders im Consumer Bereich, sind derzeit allerdings nicht abzusehen.

Literatur

[1] J. E. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, 2005.

[2] D. A. Patterson and D. R. Ditzel, "The case for the reduced instruction set computer," *SIGARCH Comput. Archit. News*, vol. 8, no. 6, pp. 25–33, 1980.

[3] J. Dehnert, B. Grant, J. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson, "The Transmeta Code Morphing Software: using speculation, recovery, and adaptive retranslation to address real-life challenges," *Code Generation and Optimization, 2003. CGO 2003. International Symposium on*, pp. 15–24, March 2003.

[4] A. Klaiber, "The Technology Behind Crusoe Processors (Whitepaper)," 2000.

[5] T. Boku, M. Sato, D. Takahashi, H. Nakashima, H. Nakamura, S. Matsuoka, and Y. Hotta, "Megaproto/e: power-aware high-performance cluster with commodity technology," *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp. 8 pp.–, April 2006.

[6] Q. Schmierer and A. Wottreng, "IBM AS/400 processor architecture and design methodology," in *Computer Design: VLSI in Computers and Processors, 1991. ICCD 91. Proceedings, 1991 IEEE International Conference on*, Oct 1991, pp. 440–443.

[7] S. Hu and J. Smith, "Using dynamic binary translation to fuse dependent instructions," *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*, pp. 213–224, March 2004.

[8] L. Geppert and T. Perry, "Transmetas magic show [microprocessor chips]," *Spectrum, IEEE*, vol. 37, no. 5, pp. 26–33, May 2000.

[9] J. C. Dehnert, "The Transmeta Crusoe: VLIW Embedded in CISC (Slides)," *7th International Workshop, SCOPES 2003, Vienna, Austria, September 24-26, 2003, Proceedings*, 2003.

[10] J. Borkenhagen, G. Handlogten, J. Irish, and S. Levenstein, "AS/400 64-bit powerPC-compatible processor implementation," in *Computer Design: VLSI in Computers and Processors, 1994. ICCD 94. Proceedings., IEEE International Conference on*, Oct 1994, pp. 192–196.