Protokollimplementierung mit Integrated Layer Processing

Torsten Braun

Institut National de Recherche en Informatique et en Automatique (INRIA), Sophia Antipolis

1 Einführung

In den letzten Jahren stieg die Verarbeitungsleistung moderner RISC-Prozessoren in einem stärkeren Maße als die Zugriffsgeschwindigkeit auf den Hauptspeicher an. Insbesondere in Kommunikationsarchitekturen mit zahlreichen Datenmanipulationsfunktionen, d.h. Funktionen, die lesend oder schreibend auf Benutzerdaten zugreifen, wird der Zugriff auf den Hauptspeicher zum wesentlichen Engpaß. Um diesen Engpaß zu beseitigen, kann zunächst versucht werden, Hauptspeicherzugriffe durch Eliminierung von Kopieroperationen zu beseitigen. Es ist offensichtlich, daß dieser Ansatz nur begrenzt zum Erfolg führen kann. Eine weitere Möglichkeit besteht darin, parallele Implementierungsarchitekturen zu verwenden, und beispielsweise für die Verarbeitung von Benutzer- und Kontrolldaten sowie für den Sende- und Empfangsteil separate Verarbeitungseinheiten einzusetzen. [BS94, Bra94a, BSSZ94, Bra95a, Bra94b, BSZ94b, BSZ94a]. Ein weiterer möglicher Ansatz besteht in der Verwendung sogenannter Cache-Speicher. Cache-Speicher sind um ein Vielfaches schneller als der Hauptspeicher, in der Regel ist der Cache-Zugriff kaum langsamer als auf die Prozessor-Register. Dadurch vermeiden sie Wartezyklen, die in der Regel beim Zugriff auf den Hauptspeicher erforderlich sind. Die Cache-Speicher sind entweder direkt auf der CPU (on-chip/first-level Cache) oder extern (off-chip/second-level Cache) untergebracht. Nachteil bei den Cache-Speichern sind derzeit die deutlich höheren Kosten und ihre begrenzte Größe. Beispielsweise beträgt die Größe des first-level Cache bei einem Sun SuperSPARC Prozessor 16 kbyte für Daten und 20 kbyte für Instruktionen, beim DEC Alpha 21064 Prozessor sind beide first-level Caches lediglich 8 kbyte groß. Aufgrund der begrenzten Größe ist es deshalb entscheidend, daß immer nur die benötigten Daten im Cache-Speicher vorhanden sind. Insbesondere, wenn verschiedene Teile eines Kommunikationssystems (z.B. Ebenen) in unterschiedlichen Prozessen oder Prozeduren implementiert sind, ist die Wahrscheinlichkeit, daß die benötigten Daten während der gesamten Verarbeitung im Kommunikationssystem immer im Cache liegen, geringer als bei einer integrierten Implementierung, bei der gegebenfalls mehrere Ebenen in einer Prozedur zusammengefaßt werden.

2 Integrated Layer Processing und Application Layer Framing

Integrated Layer Processing (ILP) ist ein Implementierungskonzept, welches dem Implementierer von Kommunikationsprotokollen erlaubt, verschiedene Datenmanipulationen in möglichst wenigen, im besten Fall einer oder zwei Verarbeitungsschleifen (ILP loop) zusammenzufassen. Dadurch sollen zum einen unnötige Datenkopieroperationen eliminiert werden, und zum anderen soll dadurch die Wahrscheinlichkeit erhöht werden, daß sich die Daten während der Verarbeitung durch die einzelnen Datenmanipulationsfunktionen des Kommunikationsytems immer im Cache befinden.

Idealerweise werden bei einer ILP loop die Daten einmal vom Hauptspeicher gelesen, die erforderlichen Datenmanipulationsfunktionen sämtlicher Ebenen werden auf der gelesenen Einheit (in der Regel wenige Bytes, z.B. 4/8 Bytes) ausgeführt und das Ergebnis wird schließlich in den Zielspeicher geschrieben. Sämtliche Datenmanipulationsfunktionen arbeiten dabei auf Registern oder dem Cache-Speicher.

Im Gegensatz dazu wird bei einer nicht-integrierten Implementierung bei jeder Datenmanipulationsfunktion das gesamte Paket gelesen, bearbeitet und schließlich wieder in den Speicher zurückgeschrieben. Gegebenenfalls werden die Pakete zur Übergabe zwischen den Funktionen von einem Speicher in einen anderen kopiert. Die Kopiervorgänge lassen sich aber in der Regel eliminieren, wenn die verschiedenen Funktionen im gleichen Adreßraum implementiert sind. Außerdem wurde in Experimenten festgestellt, daß sich auch bei einer nicht-integrierten Implementierung zumindest für nicht zu große Pakete ein zufriedenstellendes Cache-Verhalten erreichen läßt, sofern man entsprechend dem Application Layer Framing (ALF) Konzept die Paketgrenzen bei der Protokollverarbeitung nicht durch Multiplex- oder Segmentierfunktionen verändert [BD95]. ALF bedeutet im wesentlichen, daß die Anwendung die Benutzerdaten in für sie passende Einheiten (application data units, ADUs) unterteilt. Diese ADUs werden im gesamten Kommunikationssystem nicht weiter zerteilt und werden nicht mit anderen Einheiten wieder kombiniert. Dadurch werden auch aufwendige Pufferoperationen in Warteschlangen bei der Übergabe der Daten zwischen den Ebenen vermieden. Nicht zuletzt deshalb ist die Wahrscheinlichkeit, daß die Daten im Cache-Speicher beleiben, relativ groß. Vorausgesetzt, daß die Größe der Cache-Speicher zukünftig wieder zunimmt, erscheint ALF auch ohne ILP ein interessantes Konzept zu sein.

In einem einfachen Beispiel wurde durch die Anwendung des ILP-Konzepts bei der Integration von Marshalling und Prüfsummenberechnung eine Leistungssteigerung von 40 Prozent erreicht.

ALF wurde zusammen mit ILP von Clark und Tennenhouse entwickelt [CT90]. Auf diesen beiden Konzepten basierende Kommunikationssysteme werden in dem europäisch australischen Forschungsprojekt HIPPARCH entwickelt. An diesem Projekt sind neben INRIA auch SICS, University College London und die University of Technology Sydney beteiligt [CH94].

3 Probleme und Nachteile von ILP

ILP kann nur unter bestimmten Bedingungen und Voraussetzungen sinnvoll eingesetzt werden. In einer ILP loop dürfen keine Operationen vorkommen, die PDUs zerteilen oder kombinieren, d.h. das ALF-Konzept ist eine grundlegende Bedignung für ILP. Allerdings ist das Vermeiden von Multiplexen nicht nur für ILP sondern auch für andere effizienzsteigernde Implementierungstechniken und für die Einhaltung von verbindungsspezifischen Dienstgten eine Notwendigkeit.

Eine andere Beschränkung ist, daß in einer ILP loop nur sogenannte 'ordering constrained' Protokollfunktionen angewendet werden können. Protokollfunktionen dieser Art erfordern, daß die Verarbeitung einer Protokollfunktion seriell vom Beginn zum Ende eines Pakets ablaufen muß. Beispiele solcher Funktionen sind CRC-Prüfsummen oder Stream-Cipher-Verschlüsselungsalgorithmen. Die TCP-Prüfsumme oder Block-Ciper-Verschlüsselungsalgorithmen sind Gegenbeispiele, d.h. sie sind gut für ILP geeignet. Es ist aber auch bei den nicht 'ordering constrained' Protokollfunktionen notwendig, daß die Größe der Header a priori bekannt ist, da die Datenmanipulationsoperationen auf bestimmte Grenzen (z.B. 8 Bytes für Verschlüsselungsalgorithmen oder 2 Bytes für die TCP-Prüfsumme ausgerichtet sein müssen.

Ein weiteres Problem besteht darin, daß Datenmanipulationsfunktionen den Header-Inhalt beinflussen. Beispielsweise wird das Ergebnis der TCP-Prüfsummenberechnung in den TCP-Header geschrieben. Längenfelder, welche die Länge der Benutzerdaten anzeigen, sind weitere Beispiele. Die Abhängigkeiten zwischen Daten und Header verhindern, daß zunächst der Header vollständig verarbeitet und danach die Benutzerdaten. Ansätze zur Lösung dieses Problems werden in [BD95, AL93] vorgeschlagen.

Weitere ILP-Nachteile bestehen in der begrenzten Möglichkeit, eine ILP-Implementierung flexibel zu gestalten. Eine effiziente ILP-Implementierung erfordert die Benutzung von Macro-Inlining an Stelle von Funktionsaufrufen. Funktionsaufrufe würden den Performance-Gewinn durch ILP wieder aufheben. Dies bedeutet, daß die Art der verwendeten Protokollfunktionen zur Übersetzungszeit bekannt sein muß und somit eine dynamische Rekonfiguration des Kommunikationssystems aus Leistungsgründen nur in einem unbefriedigendem Maß möglich ist.

Weitere Probleme bestehen in Modularitätsaspketen, der unterschiedlichen Verarbeitungseinheiten der verschiedenen Datenmanipulationsfunktionen und den Anhängigkeiten zwischen Kontroll- und Datenteil innerhalb eines Kommunikationssystems. Lösungsansätze werden in [BD95, Bra95b, AL93] diskutiert.

4 ILP-Implementierungsexperiment

Um den erzielbaren Nutzen durch ILP abschätzen zu können, wurde ein umfangreicheres Experiment in einer realen Workstation-Umgebung durchgeführt.

4.1 Ein ILP-Kommunikationssystem basierend auf TCP/IP

Als Beispielapplikation wurde ein einfacher Dateitransfer mit Verschlüsselungsfunktion über einer TCP-Implementierung im User Space implementiert. TCP wurde als linkbare Bibliothek realisiert, d.h. eine Anwendung (z.B. Audio- oder Videoanwendungen [BZ94]) muß die TCP-Bibliothek dazubinden [BDH95].

Beim Senden werden die Benutzerdaten nach dem Marshalling verschlüsselt und in den Übertragungswiederholungspuffer von TCP kopiert. Danach wird die TCP Prüfsumme berechnet und schließlich werden die Daten in den Kern-Puffer übertragen und von IP weiterverarbeitet.

Empfangene IP-Pakete werden von einem TCP-Kern-Teil an den korrekten Anwendungsprozeß weitergeleitet. Der TCP-Kern-Teil führt damit im wesentlichen Demultiplex-Funktionen aus und ist mit

UDP vergleichbar. Die Unterschiede bestehen im Fehlen eines speziellen Headers und in der Prüfsummenberechnung. Die Daten des TCP-Paketes wird direkt in den von der Anwendung gewünschten Speicher geschrieben, der Header wird in einen TCP-internen Puffer kopiert. Dieses Vorgehen ist nur dann möglich, wenn das TCP-Paket die Bedingungen des Header Prediction Tests erfüllt, d.h. z.B. der Header die erwartete Länge (keine Optionen) besitzt und die Sequenznummer mit der erwarteten übereinstimmt. Danach wird die Prüfsumme getestet, die Daten entschlüsselt und die Nachricht über eine Unmarshalling-Routine an die Anwendung übergeben.

Zur Erzeugung der Marshalling/Unmarshalling-Routinen wurde der MAVROS-Stub-Compiler [Hui90] verwendet, der aus einer ASN.1-Beschreibung der Datentypen Routinen zur Transformation in das XDR Format erzeugt. Als Verschlüsselungsalgorithmus diente eine vereinfachte Version des SAFERK-64 Algorithmus [Mas94]. Dieser Verschlüsselungsalgorithmus soll besonders leistungsfähige Software-Implementierungen erlauben.

Zur Erzeugung der ILP-Implementierung wurde auf der Sendeseite Marshalling, Verschlüsselung, Kopieroperation und Prüfsummenberechnung integriert, indem die Marshalling-Routinen um entsprechende Macro-Anweisungen erweitert wurden. D.h. die aufgerufene Marshalling-Routine schreibt die Daten bereits verschlüsselt in den TCP-Übertragungswiederholungspuffer. Beim Empfangen wurde die Prüfsummenberechung, Entschlüsselung und Unmarshalling kombiniert.

Eine automatische Erzeugung von ILP-Marshalling-Routinen wurde für den INRIA Stub Compiler (ISC) entwickelt. ISC verwendet zur Beschreibung der Datentypen die Programmiersprache C mit Annotationen, was insbesondere die Erstellung verteilter Anwendungen erleichtert.

4.2 Leistungsbewertung

Die ILP-Implementierung wurde in Leistungsmessungen mit den nicht-integrierten Versionen mit TCP im User Space und im Kernel Space verglichen. Die Experimente wurden auf verschiedenen SUN SPARCstations und DEC-AXP Workstations durchgeführt. Außerdem wurde mit verschiedenen Simulationswerkzeugen das Cache-Verhalten der Implementierung analysiert.

Leistungsmessungen Die Meßergebnisse zeigten bei der Dateitransfer-Applikation Leistungsvorteile von ILP im Bereich von 20 Prozent bei den Paketverarbeitungszeiten, d.h. der Zeit, welche die Verarbeitung eines Pakets zwischen Anwendung und Kern innerhalb des implementierten Kommunikationssubsystems erforderlich ist. Aufgrund des System-Overheads reduzierte sich der Leistungsgewinn beim Durchsatz auf ca. 10 Prozent. Allerdings wurde mit der nicht-integrierten Version mit TCP im Kern ein etwas höherer Durchsatz erzielt, obwohl auch hier die Paketverarbeitungszeiten höher waren als bei der ILP-Implementierung. Der Grund für den besseren Kern-Durchsatz ist in der niedrigeren Anzahl von Prozeßwechseln zu suchen, da bei der User-Level-Implementierung für jedes TCP-Paket (auch für reine Quittungen) ein Prozeßwechsel stattfindet. Vorschläge zur Umgehung dieses Problems sind in [Bra95b] zu finden.

Einen wesentlichen Einfluß auf die Ergebnisse hat auch die verwendete Rechnerplattform. So machte sich abhängig vom Betriebssystem der Nutzen von ILP sehr unterschiedlich bemerkbar. Bei effizienten Betriebssystemen wie Solaris war der System-Overhead wesentlich geringer als beispielsweise auf einer DEC-AXP Workstation mit OSF/1, was natürlich den Nutzen von ILP deutlicher erkennen ließ. Mit fortgeschrittenen Netzwerkadaptern (z.B. single-copy-Adapter) und deren effizienten Integration in die Systemumgebung werden die ILP-Vorteile noch deutlicher sichtbar werden.

In einem weiteren Experiment wurde der Verschlüsselungsalgorithmus durch einen sehr einfachen Algorithmus ersetzt. Dadurch stieg die Verbesserung des Durchsatzes durch ILP auf 15 Prozent bei einer Verbesserung der Paketverarbeitungszeiten um 35 Prozent. D.h. die Komplexität der Datenmanipulationsfunktionen beeinflußt den erzielbaren Gewinn spürbar. Bei komplexen Datenmanipulationen dominiert zwangsläufig die CPU-Geschwindigkeit und der langsame Speicherzugriff macht sich weniger stark bemerkbar. Hier scheint auf den ersten Blick ILP nicht allzu große Vorteile bringen, bei einer weiteren stärkeren Zunahme der CPU-Geschwindigkeit im Vergleich zur Speicherzugriffsgeschwindigkeit wird ILP jedoch zunehmend interessanter.

Simulation des Cache-Verhaltens Das Speicherzugriffs- und Cache-Verhalten der vermessenen Programme wurde mit verschiedenen Simulationswerkzeugen untersucht. Es zeigte sich dabei, daß die erzielten Leistungssteigerungen durch ILP eher auf der reduzierten Anzahl von Speicherzugriffen als auf einer Erhöhung der Cache-Hit-Rate basieren. Die eingesparten Speicherzugriffe entsprachen beim Senden und Empfangen von Daten 3-4 Kopieroperationen. In den meisten Fällen ging durch ILP zwar auch die absolute Anzahl von Cache misses zurück, doch der relative Anteil der Cache misses war bei der nicht-integrierten Implementierung höher. Dieser zunächst überraschende Effekt hat zwei Ursachen: Zum einen lassen sich bestimmte Cache misses, wie das erstmalige Lesen und das letztmalige Schreiben in den Zielspeicher nicht vermeiden. Dies Operationen führen sehr häufig zu Cache misses. Der relative Anteil dieser Operationen an der Gesamtzahl der Speicherzugriffe erhöht sich bei der ILP-Implementierung. Der zweite Grund liegt in der Art der ausgewählten Datenmanipulationsfunktionen, im speziellen Beispiel am Verschlüsselungsalgorithmus. Der Algorithmus benötigt mehrere Tabellen zur Verschlüsselung, greift also nicht nur auf die Benutzerdaten während der Manipulationsoperationen zu. Bei der ILP-Implementierung wird die Verschlüsselung immer wieder durch andere Manipulationen unterbrochen und damit die Wahrscheinlichkeit, daß sich die benötigten Daten im Cache bleiben, reduziert. Es muß allerdings betont werden, daß die nicht-integrierte Implementierung derart gestaltet wurde, daß die Datenmanipulationsfunktionen dicht aufeinanderfolgen. Eine hohe Cache-Hit-Rate wird nicht zuletzt dadurch erreicht, daß auch die nicht-integrierte Implementierung nach dem ALF-Konzept realisiert wurde. Die Ergebnisse auf der DEC-AXP-Workstation bezüglich des Daten-Cache waren den SUN-Ergebnissen sehr ähnlich. Ein weiterer Nachteil war bei der ILP-Implementierung auf DEC-AXP zu sehen. Durch den extrem kleinen Instruktions-Cache verursachte die ILP-Implementierung deutlich mehr Cache misses. Eine detaillierte Darstellung der Ergebnisse findet sich in [BD95].

5 Diskussion und Ausblick

Das durchgeührte Experiment hat deutlich die Grenzen von ILP aufgezeigt, die das Konzept beim Einsatz in herkömmlichen Protokollen und in üblichen Implementierungsumgebungen mit sich bringt. Zwar wird ein gewisser Leistungszuwachs durch ILP erreicht, aber aufgrund der damit verbundenen Nachteile (nur gewisse Protokollfunktionen erlauben ILP, begrenzte Flexibilität, aufwendigere Implementierung) erscheint es fraglich, ob sich eine ILP-Implementierung lohnt. Die bloße Anwendung des ALF-Konzepts erscheint zunächst eine gute Alternative zu sein. Speziell bei leistungsfähigeren Workstations mit superschnellen CPUs und single-copy-Netzwerkadaptern erscheint ILP jedoch interessanter zu werden. Auch können verbesserte Protokollarchitekturen durch feste Header-Formate, auf die Basiseinheiten der Datenmanipulationsfunktionen abgestimmte Header-Formate, Demultiplex-Unterstützung durch eindeutige Verbindungsidentifikatoren sowie getrennte Kontroll- und Datenpakete ILP-Implementierungen wesentlich vereinfachen.

Auch erscheint ILP für Kommunikationssysteme mit aufwendigen Datenmanipulationen interessant, falls die teueren Datenmanipualationen durch programmierbare Hardware verarbeitet werden können. Oft werden nämlich für aufwendige Datenmanipulationsfunktionen Spezial-Hardware-Lösungen eingesetzt. Falls in einem Kommunikationssubsystem mehrere solcher Spezial-Hardware-Bausteine genutzt werden, sind mehrere Datentransfers vom Speicher zu diesen Spezial-Hardware-Bausteinen und zurück erforderlich. Wie in [Bra95b] vorgeschlagen, könnten die aufwendigen Datenmanipulationsfunktionen zu einer ILP loop zusammengefaßt und durch Anwendung des ILP-Ansatzes auf einem einzigen programmierbaren Baustein implementiert werden. Es bietet sich hier zur automatisierten Implementierung an etwas einen ILP-Stub-Compiler zu realisieren, der nicht wie die ISC-Erweiterung C-Funktionen sondern etwa VHDL-Funktionen generiert.

Literatur

[AL93] M. B. Abbott und Peterson L. L. Increasing Network Throughput by Integrated layer Processing. In *IEEE/ACM Transcations on Networking, Vol. 1, No. 5*, Oktober 1993, Seite 600–610.

[BD95] Torsten Braun und Christophe Diot. Protocol Implementation Using Integrated Layer Processing. *zur Veröffentlichung eingereicht*, Januar 1995.

[BDH95] Torsten Braun, Christophe Diot und Anna Hoglander. An Experimental User-Level Implementation of TCP. *interner Bericht*, 1995.

[Bra94a] Torsten Braun. Architektur, Implementierung und Leistungsbewertung eines parallelen Transportsubsystems fr zellenbasierte Hochgeschwindigkeitsnetze. In *GI/ITG-Fachgruppe 'Kommunikation und Verteilte Systeme': 1. Arbeitstreffen 'Architektur und Implementierung von Hochleistungs-Kommunikationssystemen'*, Karlsruhe, 17./18. Januar 1994.

[Bra94b] Torsten Braun. PATROCLOS: A Flexible and High-Performance Transport Subsystem. In *4th IFIP Workshop on Protocols for High-Speed Networks*, Vancouver, 10.-12. August 1994. Seite 183–201.

[Bra95a] Torsten Braun. Ein paralleles Transportsubsystem für zellenbasierte Hochgeschwindigkeitsnetze. In *Kommunikation in Verteilten Systemen*, Informatik aktuell, Chemnitz, 22.24. Februar 1995. Springer-Verlag, Seite 562–566.

[Bra95b] Torsten Braun. Limitations and Experiences of Integrated Layer Processing. In *1. Gemeinsame Jahrestagung der Gesellschaft für Informatik (GI) und der Schweizer Gesellschaft für Informatik (SI)*, Zürich, 18.-20. September 1995.

[BS94] Torsten Braun und Claudia Schmidt. A Parallel Transport Subsystem Implementation for High Performance Communication. Concurrency: Practice and Experience, Special Issue on High Performance Distributed Computing, Juni 1994.

[BSSZ94] Torsten Braun, Claudia Schmidt, Burkhard Stiller und Martina Zitterbart. A Protocol Architecture for a Flexible High Performance Communication Subsystem. In *CaberNet Workshop*, Dublin, 23.-25. März 1994.

[BSZ94a] Torsten Braun, Jochen Schiller und Martina Zitterbart. Implementation of Transport Protocols Using Parallelism and VLSI Components. In *International Telecommunication Symposium 1994 (ITS'94*, Rio de Janeiro, 22.-25. August 1994. Seite 285–290.

[BSZ94b] Torsten Braun, Jochen Schiller und Martina Zitterbart. A Modular VLSI Implementation Architecture for Communication Subsystems. In *4th IFIP Workshop on Protocols for High-Speed Networks*, Vancouver, 10.-12. August 1994. Seite 119–134.

[BZ94] Torsten Braun und Martina Zitterbart. Videokonferenzen im Internet. In *Praxis der Informationsverarbeitung und Kommunikation (PIK) 17, 4/1994*, 1994, Seite 253–254.

[CH94] Jon Crowcroft und Christian Huitema. *1st International Workshop on High Performance Protocol Architectures (HIPPARCH'94)*. Sophia-Antipolis. 15.-16. Dezember 1994.

[CT90] David Clark und Dave Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM*, September 1990, Seite 200–208.

[Hui90] Christian Huitema. MAVROS: Highlights on an ASN.1 Compiler. *interner Bericht*, 1990.

[Mas94] John Massey. SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm. In *Lecture Notes in Computer Science 809*, Berlin, Heidelberg, 1994. Seite 1–17.