



Fortgeschrittenenpraktikum

**Empirische Identifikation von  
Parametern mit Einfluss auf die  
Effizienz von Virtualisierung  
– am Beispiel VMware ESXi**

Günter Lemberger



INSTITUT FÜR INFORMATIK  
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



Fortgeschrittenenpraktikum

**Empirische Identifikation von  
Parametern mit Einfluss auf die  
Effizienz von Virtualisierung  
– am Beispiel VMware ESXi**

Günter Lemberger

Aufgabensteller: Prof. Dr. Dieter Kranzlmüller

Betreuer: Dr. Nils gentschen Felde  
Dr. Tobias Lindinger

Abgabetermin: 05. August 2010



Hiermit versichere ich, dass ich das vorliegende Fortgeschrittenenpraktikum selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 05. August 2010

.....  
*(Unterschrift des Kandidaten)*



## Abstract

Virtualisierung wird zur Zeit immer wichtiger, vor allem von Servern. So werden häufig in Rechenzentren oder in kleineren Serverräumen vermehrt auf virtuelle Lösungen zurückgegriffen. Durch den Einsatz solcher Systeme können vereinzelt physische Komponenten eingespart oder sogar komplett auf ganze Systeme verzichtet werden. Jedoch ist zu bedenken, dass eine virtuelle Lösung nicht zwingend den gleichen Durchsatz wie ein physisches System hat und es dadurch zu einem Verlust kommen könnte, was in dieser Arbeit näher betrachtet und untersucht wird. Virtualisierte Lösungen haben einen Einfluss auf die Antwortzeiten und die Datenraten der verschiedenen Komponenten und dadurch auch deren Leistungsfähigkeit. Es ist aber auch durchaus möglich, dass durch einen geschickten Einsatz solcher Lösungen eine insgesamt deutlich bessere Auslastung der Komponenten erreicht wird.

Der Einfluss durch den Virtualisierer und die Leistungsfähigkeit wird in dieser Arbeit mit Hilfe verschiedener Tests unter Verwendung des Virtualisierers VMware getestet und ausgewertet. Aus diesem Grund, und der Tatsache, dass es unzählige Virtualisierungslösungen gibt, und eine vollständige Untersuchung den Rahmen dieser Arbeit bei weitem sprengen würde, wurde der Augenmerk alleine auf VMware gelegt.

An dieser Stelle sei erwähnt, dass es drei weitere Arbeiten unter identischen Hardware- und Softwarevoraussetzungen von Eva Tsoiprou, Bastian Gebhardt und Anton Romanjuk gibt, die die Virtualisierer OpenVZ/Virtuozzo, XEN und Microsoft Hyper-V näher betrachten.

In dieser, wie in jeder der erwähnten drei, wird im ersten Schritt der reine Verlust der Virtualisierung bestimmt, um dann im zweiten Schritt die Leistung unter Mehr-VM-Betrieb zu evaluieren.





# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation und Ziele . . . . .	1
1.2	Vorgehensweise . . . . .	1
<b>2</b>	<b>Theorie und Grundlagen</b>	<b>3</b>
2.1	Virtualisierung . . . . .	3
2.2	VMware ESXi 3.5 . . . . .	5
2.2.1	CPU . . . . .	6
2.2.2	RAM . . . . .	6
2.2.3	Virtualisierung der Peripheriegeräte . . . . .	6
2.2.4	iSCSI . . . . .	7
2.2.5	AMD-V . . . . .	8
2.2.6	Dual-Channel . . . . .	8
2.3	Zeitgebung in Virtuellen Maschinen . . . . .	8
2.3.1	Problematik . . . . .	9
2.3.2	Lösungsansätze . . . . .	9
<b>3</b>	<b>Aufbau, Konfiguration und Anpassung</b>	<b>11</b>
3.1	Aufbau und Konfiguration der Systeme . . . . .	11
3.1.1	Hardware . . . . .	11
3.1.2	Infrastruktur . . . . .	12
3.2	Anpassung der Benchmarktools . . . . .	13
3.2.1	CPU . . . . .	13
3.2.2	RAM . . . . .	14
3.2.3	Netz und Disk . . . . .	16
<b>4</b>	<b>Durchführung und Auswertung der Tests</b>	<b>19</b>
4.1	Synthetische Benchmarks . . . . .	19
4.1.1	Durchführung . . . . .	19
4.1.2	Auswertung . . . . .	22
4.2	Parallele Benchmarks . . . . .	29
4.2.1	Durchführung . . . . .	29
4.2.2	Auswertung . . . . .	30
<b>5</b>	<b>Fazit und Ausblick</b>	<b>37</b>
5.1	Erreichte Ziele . . . . .	37
5.2	Fazit . . . . .	37
5.3	Ausblick . . . . .	37
	<b>Abbildungsverzeichnis</b>	<b>39</b>

*Inhaltsverzeichnis*

<b>Tabellenverzeichnis</b>	<b>41</b>
<b>Ergebnisse</b>	<b>43</b>
<b>Quelltexte und Patches</b>	<b>49</b>
<b>Literaturverzeichnis</b>	<b>85</b>

# 1 Einleitung

In der heutigen Zeit werden immer häufiger eine große Anzahl an Rechnern benötigt. Während vor einiger Zeit jeder Serverrechner noch durch einen physischen Rechner dargestellt wurde, werden heute vermehrt alle Server durch virtuelle Maschinen realisiert. Eine virtuelle Maschine, kurz VM, ist ein virtueller Computer, der nicht aus physischer Hardware, sondern nur aus Software besteht. So können mehrere virtuelle Maschinen auf einem physischen Rechner betrieben werden. Durch einen geschickten Einsatz von virtuellen Maschinen können Ressourcen gespart und Systeme effizienter genutzt werden, weshalb der Einsatz von virtuellen Maschinen immer wichtiger wird und oft im Fokus wissenschaftlicher Arbeiten steht. Vor allem aus wirtschaftlicher Sicht ist ein Einsatz von virtuellen Systemen oftmals sinnvoll[vmwc]. Die Verteilung der Ressourcen wird von einem 'Hypervisor' verwaltet. Er verteilt die Hardware Ressourcen so, dass bei Bedarf für jede einzelne VM bzw. deren Betriebssystem, die benötigte Ressource zur Verfügung steht.

Der Einsatz von solchen Systemen hat aber nicht nur Vorteile. So geht man davon aus, dass die Ausführungen von Programmen in einer VM teils erheblich langsamer sind als auf nativen Systemen. Vor allem beim Betrieb von mehreren virtuellen Maschinen gleichzeitig kann es sein, dass durch das ständige Umschalten zwischen den VMs Ressourcen vergeudet werden[ibm05].

## 1.1 Motivation und Ziele

Bei Virtualisierung erhält man technisch bedingt einen Overhead durch das Verwalten der virtuellen Maschinen. Die Hardwareelemente müssen in VMs nachgebildet werden und gegebenenfalls auch an die physische Hardware des Virtualisierers weitergeleitet werden. Laufen mehrere VMs gleichzeitig auf einem Virtualisierer, muss der Fokus durch einen Ressourcenscheduler im Wechsel an die verschiedenen VMs gegeben werden. Dieser 'Verlust' ist in vielen Fällen sehr interessant, weil man durch die Kenntnis den Einsatz von virtuellen Maschinen effizienter gestalten kann. Auch durch spezifische Konfigurationen kann das physische System besser genutzt werden. Das Ziel dieser Arbeit ist es ein virtuelles System mit einem nativen System unter Verwendung verschiedener Parameter zu vergleichen und die Performanz dieser zu evaluieren. Zuerst werden die einzelnen Komponenten und auch nur einzelne virtuelle Maschinen mit nativen Systemen verglichen um dadurch einen Wirkungsgrad zu erhalten. Mit Hilfe der Ergebnisse werden mehrere VMs parallel getestet um so die Belastbarkeit des Hypervisors und den Verlust des Scheduling zu bestimmen zu können.

## 1.2 Vorgehensweise

In Kapitel 2 werden zuerst die verschiedenen Typen von Hypervisor vorgestellt. Danach wird auf den hier speziell behandelten Virtualisierer VMware ESXi genauer eingegangen. Hierzu werden die Funktionsweisen und die spezielle Umsetzung der Komponenten CPU, Arbeits-

## 1 Einleitung

speicher, Netzwerk und Festplatte näher erläutert. Anschließend wird noch die Problematik der Zeitgebung in einer virtuellen Maschine näher betrachtet.

Im folgenden Kapitel 3 wird das hier extra dafür aufgebaute Szenario beschrieben. Welche Komponenten wurden verwendet, was wurde installiert, welche Tests wurden wie durchgeführt. Im zweiten Teil dieses Kapitels wird noch näher auf die Anpassung der Benchmarktools auf die gegebenen Anforderungen eingegangen.

Im vorletzten Kapitel 4 geht es um die Durchführung der Test und die Auswertung dieser. Die Testspezifikationen werden aufgelistet und erklärt. Zuerst werden die synthetischen Tests beschrieben und die Ergebnisse erläutert, da sich die parallelen Tests aus den synthetischen abgeleitet haben. Bei den synthetischen Tests handelt es sich um komplett künstliche Tests, da diese nur eine VM testen und darin auch nur eine einzelne Komponente. Aus den erhaltenen Ergebnissen wurde dann eine etwas realistischere Testsituation geschaffen und diese durchgeführt. Im 5 und letzten Kapitel wird ein kurzes Fazit über die Beobachtungen und die erzielten Ergebnisse geschlossen und ein Ausblick über weitere auf diese Arbeit aufbauende Arbeiten gegeben.

## 2 Theorie und Grundlagen

In diesem Kapitel werden die technischen Grundlagen, die verschiedenen Ansätze und Typen von Virtualisierung und ein paar Begriffe näher erläutert. Danach werden noch ein paar Details zum hier verwendeten Virtualisierer VMware ESXi 3.5 erklärt und näher betrachtet.

### 2.1 Virtualisierung

Der Einsatz von virtuellen Systemen soll die Effizienz von IT Ressourcen steigern. Das alte "one server, one application" Modell sollte ersetzt werden und mehrere virtuelle Maschinen auf einer physischen Maschine laufen. Eine virtuelle Maschine, kurz VM, ist ein virtueller Computer, der nicht aus Hardware sondern nur aus Software besteht [vmwb][itwb]. Diese wird also nicht direkt auf die Hardware installiert, sondern auf eine Zwischenschicht aufgesetzt. Das geschieht, indem man das normale Ringmodell der X86-Architektur verändert. Das Ringmodell besteht aus 4 Ringen. Die Ringe beschreiben die Privilegierungs- und Sicherheitsstufen. Der erste oder innerste beinhaltet den Betriebssystemkernel sowie die Hardwaretreiber. Nach außen hin werden die Ringe immer unprivilegierter und die Anwendungen befinden sich im äußersten und damit vierten Ring. Dieses Modell soll verhindern, dass Anwendungen unerlaubte Speicherzugriffe erhalten, indem sie über den unprivilegierten Ring nur indirekt auf die Hardware zugreifen können. Die beiden Ringe 1 und 2 werden bei nativen Systemen nicht genutzt. Die Virtualisierungssoftware schiebt nun das Betriebssystem einen Ring nach außen und sich selbst auf den innersten Ring, damit es die Hardwarezugriffe selbst verwalten kann [vmw07]. Eine VM ist eigentlich ein isolierter Container, auf dem das Betriebssystem und Anwendungen laufen können als wäre es ein normaler physischer Rechner. Er verhält sich exakt so und beinhaltet seine eigene Hardware, wie CPU, RAM, Festplatten, Netzwerkkarten etc..

So ist es durchaus möglich, dass weder das Betriebssystem, die Programme noch Rechner aus dem Netzwerk von sich aus einen Unterschied zwischen einer VM und einem physischen Rechner feststellen oder dieser sich nur sehr schwer feststellen lässt, obwohl solche Maschinen nur rein aus Software entstehen. Daraus ergeben sich Vor- und Nachteile gegenüber physischen Maschinen. Jedoch gibt es auch Ansätze, die bewusst darauf setzen, dass das virtuelle System auch als solches erkannt wird, um damit eine effizientere Nutzung zu erlangen [vmwc]. Durch den Betrieb von virtuellen Plattformen kann auch schneller und effektiver auf neue Anforderungen eingegangen werden. Andere Vorteile sind:

- mehrere, vor allem aber auch verschiedene, Betriebssysteme können auf einem einzelnen Rechner laufen
- effizientere Nutzung der Hardware, energieeffizientere Nutzung der Rechner, Verwaltungsaufwand reduzierbar

Bei Virtualisierung wird grundsätzlich zwischen 4 Typen unterschieden, der Voll-, Para-, Betriebssystemvirtualisierung und der Emulation.

- **Emulation** Bei der Emulation werden komplett alle Komponenten von der physischen Hardware getrennt und diese nur durch Software dargestellt. Dies hat unter anderem den Vorteil, dass auch eine andere CPU-Architektur, als die Architektur der physischen CPU, nachgebildet werden kann. Es kann so auch jede beliebige Hardware emuliert werden.
- **Vollvirtualisierung** Jedem Gastsystem wird bei der Vollvirtualisierung aus technischer Sicht eine Standard-Hardware gegeben. Prozessor- und Hauptspeichierzugriffe werden direkt, aber nicht zwingend unverändert, an die physikalische Hardware durchgereicht. Eine Anpassung der Komponenten geschieht über die Virtualisierungsschicht. Vollvirtualisiert deshalb, weil die virtualisierte Hardware vollständig von der darunterliegenden abstrahiert ist.

Durch die komplette Emulation der Hardware entsteht ein zusätzlicher Rechenaufwand, durch den Systemleistung verloren geht, auch manchmal "Virtualisierungsschwund" genannt.

Die VMs und damit im Gast installierte Betriebssysteme wissen nichts von der virtuellen Hardware, arbeiten also normal wie mit einer physikalischen Hardware zusammen und benötigen daher im einfachsten Fall keine Anpassungen.

Das Betriebssystem läuft in diesem Fall in Ring 1 und der Virtualisierer im innersten Ring. Der Zugriff auf die Hardware erfolgt deshalb über den Ring 1 zum Virtualisierer auf Ring 0[itwc].
- **Paravirtualisierung** Im Gegensatz zur Vollvirtualisierung benötigt die Paravirtualisierung eine Anpassung des Gastsystems. Dadurch "weiß" es, dass es virtualisiert läuft und über eine abstrahierte Hardwareschnittstelle mit Hilfe von Hypercalls, speziellen CPU-Befehlen, mit dem Virtualisierer kommunizieren kann. Dabei handelt es sich um Erweiterung der Binärschnittstelle, also der Kommunikation des Betriebssystems mit der Hardware. Statt wie bei normalen Systemaufrufen mit der in diesem Fall virtuellen Hardware zu kommunizieren, ist es mit Hypercalls möglich von einer VM aus direkt den Hypervisor anzusprechen.[itwa].

Die Kommunikation erfolgt also direkt von Ring 3 nach Ring 0 und überspringt so den Kernel der virtuellen Maschine, der in Ring 1 ist[DME06].Dadurch erhofft man sich eine bessere Effizienz.
- **Betriebssystemvirtualisierung** Die Betriebssystemvirtualisierung setzt auf dem Kernel des Betriebssystems auf. Jedoch wird keine extra Abstraktionsschicht in das Ringmodell zwischen Hardware und Gastsystem eingefügt. Es wird lediglich eine Laufzeitumgebung im Host-Betriebssystem virtuell in Containern zur Separierung der einzelnen Virtuellen Maschinen zur Verfügung gestellt. Als Basis dient ein Standard-Betriebssystem, darauf befindet sich die Virtualisierungsschicht, die die Abstraktionsschicht, welche die Sicherheit und Isolierung gewährleistet, beinhaltet[vir].

Beim Einsatz einer virtuellen Lösung wird eine Softwareschicht direkt auf die Hardware oder das Betriebssystem eingefügt. Diese Software, vorher auch Virtualisierer genannt, der bei der Virtualisierung in Ring 0 ist, wird als "Hypervisor" oder auch "virtual machine monitor" bezeichnet und übernimmt die Aufteilung der Hardware auf die virtuellen Maschinen

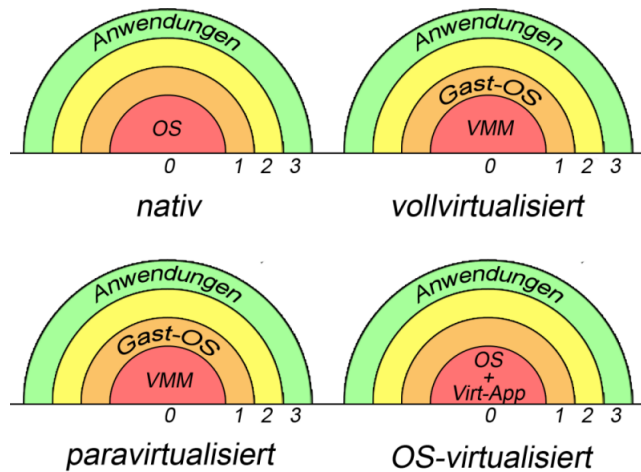


Abbildung 2.1: Ringmodell: Übersicht der Virtualisierungsmöglichkeiten

und überwacht diese. Bei den bisher vorgestellten Methoden handelt es sich um "Hypervisor" vom Typ 1, auch "hardwarebasierter Hypervisor" genannt. Im Gegensatz zu Typ 1 Hypervisor, die direkt auf die Hardware aufsetzen, benötigen die Typ 2 Hypervisor, auch hostbasierte Hypervisor genannt, ein Betriebssystem als Grundlage [Gol73]. Sie laufen also als Programm auf einem vollwertigen Betriebssystem. Dabei kommt es zu keiner strikten Abkapselung zwischen Hypervisor, Betriebssystem und virtuellen Maschinen, da der Hypervisor eine Anwendung des unmodifizierten Betriebssystems ist [Spe05][Van].

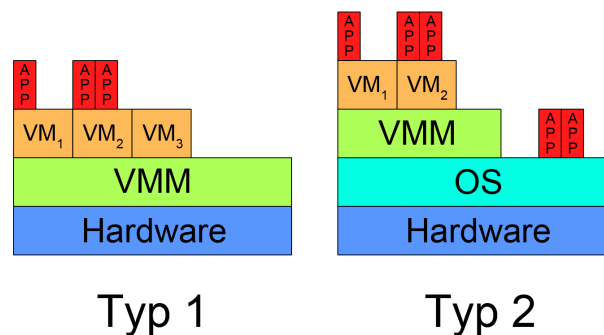


Abbildung 2.2: Vergleich der beiden Hypervisortypen

## 2.2 VMware ESXi 3.5

Die Architektur der heutigen Betriebssysteme ist so ausgelegt, dass diese direkt auf der Hardware laufen und so mit der Hardware kommunizieren können. Wie bereits in Kapitel 2.1 gezeigt, laufen sie auf dem privilegiertesten Ring 0, um somit die Hardwarezugriffe zu steuern. Die Virtualisierung erfordert also, dass eine Virtualisierungsschicht eingeschoben wird, die in Ring 0 laufen soll, um so die Hardwarezugriffe der VMs kontrollieren und steuern zu können.

Bei VMware ESXi 3.5, der hier verwendet wurde, handelt es sich um einen Vollvirtualisierer des Typs 1.

### 2.2.1 CPU

Die Schwierigkeiten bei der Virtualisierung der CPU besteht darin, dass das Betriebssystem normalerweise in Ring 0 läuft und so die Befehle auf der CPU ausführen kann. Da es nun aber in Ring 1 läuft und dadurch nicht den vollen Zugriff darauf hat, müssen manche Anweisungen eingefangen und übersetzt werden. VMware benutzt hierzu eine Kombination von “binary translation” und direkter Ausführung der Befehle. Diese Herangehensweise übersetzt nichtvirtualisierbare Anweisungen in Befehle, die von der virtuellen Hardware richtig verstanden werden. Jedoch wird der Anwendungscode, der in Ring 3 abläuft, direkt auf der CPU ausgeführt, um somit keinen Leistungsverlust hinnehmen zu müssen[vmw07].

Durch diese Methode wird eine volle Abstraktion des Gast-Betriebssystems von der Hardware erreicht. Es muss nicht angepasst werden, da sich der Hypervisor darum kümmert, dass die Betriebssystemanweisungen sofort bei Bedarf übersetzt werden und puffert diese für den weiteren Gebrauch, während die Anwendungsanweisungen unverändert und direkt ausgeführt werden[vmw05].

### 2.2.2 RAM

Ein weiterer kritischer Punkt bei der Virtualisierung ist die Verwaltung des Hauptspeichers. Dazu gehört die Zuweisung zu den VMs und die dynamische Aufteilung. Aus Anwendungssicht steht ein zusammenhängender Speicherbereich zur Verfügung, der aber nicht zwingend mit dem physischen Speicher zusammenhängend verknüpft ist. Das Gast-Betriebssystem speichert nachwievordie Abbildungen des virtuellen Speichers auf den “physischen” Speicher, der ja hier nur virtuell existiert, in Seitentabellen. Alle modernen CPUs beinhalten eine MMU (memory management unit) und einen TLB (translation lookaside buffer) um die Geschwindigkeit des virtuellen Speichers zu optimieren[vmw05].

Eine weitere Schicht für die Speichervirtualisierung ist notwendig, um mehrere VMs parallel laufen zu lassen, da diese ja nicht über einen realen physischen Speicher verfügen. Der Hypervisor kann in virtuellen Systemen nicht zulassen, dass Gastssysteme direkten Zugriff auf die MMU haben, da sich sonst gegenseitige Zugriffe auf identische Speicherbereiche nicht ausschließen lassen. Daher simuliert er für jede VM eine eigene Tabelle für Speicheradressen und konsultiert bei Anfragen selbst die MMU, um somit dem Gast-Betriebssystem den ‘normalen’ Zugriff zu erlauben. Diese neue Zwischenschicht weist dem “physischen” Speicher des Gastes einem realen Speicher zu. Um diesen Zugriff zu optimieren benutzt es zusätzlich “shadow page tables”. Diese erlauben einen direkten Zugriff auf den Speicher ohne die beiden Übersetzungsschichten. Dabei werden die “shadow page tables” immer dann aktualisiert, wenn das Gast-Betriebssystem die Zuweisung des virtuellen Speichers auf den physischen Speicher ändert[vmw07]. Dieses Vorgehen ist in Abbildung 2.3 mit einem roten Pfeil dargestellt.

### 2.2.3 Virtualisierung der Peripheriegeräte

Die restlichen Komponenten wie Eingabegeräte, Grafikkarte, Netzwerkkarten, Festplatten, etc. müssen auch noch virtualisiert werden. Dazu gehört, dass diese Komponenten zwischen den einzelnen VMs verteilt werden. Softwarebasierte Virtualisierung ermöglicht im Gegensatz zur direkten Weitergabe ein einfacheres Management und bietet mehr Möglichkeiten.



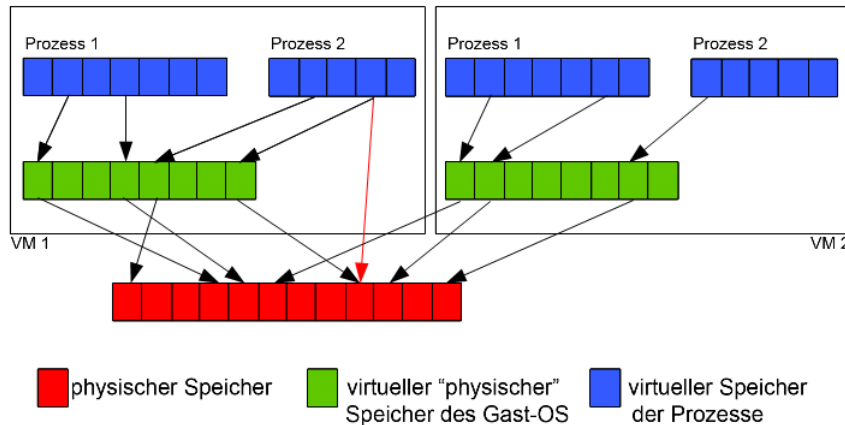


Abbildung 2.3: Virtualisierung des Hauptspeichers

Beim Netz ist es so durch den Einsatz von virtuellen Netzwerkkarten und Switches sehr leicht möglich, ein virtuelles Netz zwischen den VMs zu erstellen, ohne dass dabei der Datentransfer über die physische Hardware gelenkt werden muss[vmw05].

Der Hypervisor virtualisiert die physische Hardware für jede VM und bietet Standardkomponenten für diese an. Dadurch ist der Einsatz der gleichen VMs auf unterschiedlicher physischer Hardware problemlos möglich[vmw07].

## 2.2.4 iSCSI

iSCSI ("Internet SCSI") ist ein Protokoll, um SCSI-Festplatten über das Internet, oder in diesem Fall über LAN, also ein lokales Netz zur Verfügung zu stellen.

Dies wird ermöglicht, indem das SCSI-Protokoll zum Festplattenzugriff über TCP, getunnelt wird.

Der Grund warum iSCSI in dieser Arbeit verwendet wird ist, dass dies neben des "Fibre-Channel-Storage" die Standardmethode ist, um virtuellen Maschinen beispielsweise in Rechenzentren Festplattenspeicher zur Verfügung zu stellen. Man erhält so also einen gewissen Grad an Realität.

Man baut dabei mit dem iSCSI Client, einem sogenannten "Initiator", eine Verbindung zum Server, dem sogenannten "Target" auf. Normalerweise ist dabei eine Authentifikation, zum Beispiel über das MSCHAP Verfahren nötig.

Ist eine Verbindung zum Target hergestellt, werden die sog. "LUNs" in das System eingebunden. Dabei muss es sich nicht tatsächlich um ganze Festplatten handeln.

Diese LUNs werden allerdings nicht direkt an die VMs übergeben, sondern es werden auf diese LUNs Festplattenimages - also Dateien auf die ein Dateisystem formatiert wurde - gelegt, welche an die VMs übergeben werden.

VMware ESXi 3.5 bietet für den Einsatz von iSCSI einen Software-Initiator an, jedoch kann auch auf einen hardwarebasierten Initiator zurückgegriffen werden.

### 2.2.5 AMD-V

AMD-V ist eine Entwicklung der Firma AMD steht für AMD-Virtualization. Dabei handelt es sich um eine Befehlssatzerweiterung für AMD Prozessoren zur Optimierung von virtuellen Lösungen. Ein ähnliches Konzept, aber unabhängig voneinander entwickelt, für Intel Prozessoren bietet Intel unter dem Namen *Intel VT* an.

Diese Befehlssatzerweiterungen werden durch eine "Secure Virtual Machine" realisiert. Durch die Aktivierung dieser SVM sollte man von einer Leistungssteigerung ausgehen, welche im Kapitel 4 unter Anderem nachgeprüft wird. Es gibt durchaus Virtualisierungslösungen, die auf diese Erweiterungen aufbauen und für diese sie zwingend notwendig sind. Dies ist aber bei VMware ESXi nicht der Fall, weshalb alle Iterationen mit ein- und ausgeschalteten Erweiterungen getestet werden konnten[AMD05].

### 2.2.6 Dual-Channel

Die Fähigkeit zwei Arbeitsspeichermodule parallel zu betreiben, bezeichnet man als Dual Channel. Dadurch soll eine höhere Datentransferrate und damit eine Leistungssteigerung erzielt werden. Hierfür sind zwei getrennte Busse vom Speicherkontroller zu den Modulen nötig.

Es gibt grundsätzlich zwei verschiedene Ansätze zur Realisierung von Dual Channel, je nach Sitz des Kontrollers.

- In der klassischen Chipsatz-Architektur befindet sich der Speicherkontroller in der Northbridge, welche über den Front Side Bus an die CPU gekoppelt ist. Dadurch wird nicht die Bandbreite zwischen Prozessor und Speicher, sondern die Bandbreite zwischen Northbridge und Speicher erhöht. In dieser Architektur wird nur ein leichter Leistungsgewinn erreicht.
- Die Bitbreite des Speicherbusses und die verfügbare Speicherbandbreite wird jedoch verdoppelt, falls der Speichercontroller direkt im Hauptprozessor sitzt, was bei den neueren AMD-Modellen, seit Einführung der Athlon 64 Prozessoren, der Fall ist. Alle AMD-Prozessoren seit dem Sockel 939 unterstützen daher Dual-Channel-Betrieb. Die zum Sockel 754 zusätzlichen Pins bilden den notwendigen zweiten Speicherbus.

Im Single-Channel-Modus ist der Datenbus 64 Bit breit, im Gegensatz dazu sind dies im Dual-Channel-Modus doppelt so viele, also 128 Bit, da beide Module parallel mit 64 Bit betrieben werden können. Da die Taktrate gleich bleibt, die Daten sich aber verdoppeln, führt Dual-Channel zumindest theoretisch zur Verdoppelung des Speicherdurchsatzes, was in Kapitel 4 genauer untersucht wird.

## 2.3 Zeitgebung in Virtuellen Maschinen

Da der Host bei virtuellen Systemen dafür zuständig ist, die Zeitaufteilung der Hardware zu übernehmen, kann die virtuelle Maschine nicht die Zeit des physischen Systems exakt abbilden. Es gibt verschiedene Verfahren um dieses Problem zu beheben, jedoch kann es trotzdem zu Ungenauigkeiten in der Zeitgebung oder anderen Problemen bei manchen Programmen kommen. In diesem Abschnitt wird diese Problematik genauer erläutert und danach noch ein paar Lösungsansätze gezeigt.

### 2.3.1 Problematik

Physische Systeme messen typischerweise die Zeit auf eine der 2 folgenden Arten. Entweder mit Hilfe von 'tick counting' oder 'tickless timekeeping'. Beim 'tick counting' unterbricht eine Hardware das Betriebssystem periodisch, z.B. jede 100 Millisekunden. Das Betriebssystem zählt diese Unterbrechungen, auch ticks genannt, und kann daraus die verstrichene Zeit bestimmen. Beim 'tickless timekeeping' braucht das Betriebssystem nur den Zähler der Hardwareeinheit, welche beim Booten gestartet wird, auslesen. Das Problem der Zeitgebung in einer VM entsteht dadurch, dass die CPU nicht immer aktiv ist und so auch nicht über die "aktuelle" Zeit verfügen kann.

Es ist auch möglich die Zeit der Virtuellen Maschinen und der Hosts zu synchronisieren. Hierfür sind beim Einsatz von VMware ESXi 3.5 2 Ansätze realisierbar. Zum einen bieten die vmware-tools eine periodische Zeitsynchronisation an, zum anderen wäre eine native Synchronisation, wie z.B. `ntp`, denkbar. Beides zieht aber auch Nachteile mit sich, welche einen sinnvollen Einsatz bei den Tests nicht ermöglichen. Bei der Synchronisation der Zeit über die vmware-tools wird die Zeit periodisch synchronisiert. Die vmware-tools bemerken aber nur einen Fehler in der Zeit und setzen sie neu, wenn diese nicht der "echten" Zeit voraus ist, was bei Virtuellen Maschinen durch die Emulation verschiedener Komponenten durchaus vorkommen kann. Bei der nativen Synchronisation kann die Zeit nicht nach vorne gestellt werden, wenn sich die Virtuelle Maschine gerade in einer Unterbrechung befindet, weshalb es auch hier nicht immer zur exakten Darstellung der Zeit kommen kann[vmwa].

### 2.3.2 Lösungsansätze

Um dieses Problem mit der Zeitgebung in einer virtuellen Maschine zu übergehen, welche ja bei Benchmarks sehr wichtig ist, gibt es verschiedene Lösungsansätze. Die in der Praxis einfachste Variante ist die Zeitgebung über einen externen Server. Bei dieser Methode sendet der Client, welcher auf der zu testenden virtuellen Maschine läuft, seine Daten über das Netzwerk an einen physischen Server. Bei dieser Methode kommt es jedoch zu Ungenauigkeiten bei der Messung, da der Netzverkehr in die Berechnungen einfließt. Jedoch kann dieser Fehler bei ausreichend langen Tests vernachlässigbar klein gehalten bzw., da dieser reproduzierbar und nahezu konstant ist, herausgerechnet werden.



## 3 Aufbau, Konfiguration und Anpassung

In diesem Kapitel wird der allgemeine Aufbau des Testszenarios und die verwendeten Komponenten genauer beschrieben. Genauer eingegangen wird vor allem auf die Installation der Systeme, sowohl auf die nativen, wie auch die virtuellen und auch deren Konfiguration wird näher betrachtet. Im zweiten Teil wird die Auswahl und die Anpassung der Benchmarktools an die gegebenen Bedingungen beschrieben.

### 3.1 Aufbau und Konfiguration der Systeme

Es standen zwei identische Rechner für die Installation des Virtualisierers zur Verfügung, welche in einem lokalen Netzwerk über einen extra Rechner in die vorhandene Infrastruktur eingebunden waren. Zuerst wird die Hardware der beiden identischen Rechner näher beschrieben, anschließend die bereits kurz erwähnte Infrastruktur.

#### 3.1.1 Hardware

Als Hostrechner wurde kein besonders leistungstarker Rechner, sondern eher ein handelsüblicher Rechner, mit folgenden Komponenten verwendet:

- **CPU** AMD ATHLON64 X2 4800+ 65W AM2
- **Mainboard** ASUS M2N-SLI Deluxe nForce570SLI
- **RAM** 4 x SAMSUNG 1024MB DDR2-800 PC2-6400
- **Grafikkarte** Sapphire Radeon HD3450 256MB PCIe 2.0
- **Festplatte** Seagate Barracuda 7200.10 250 GB SATA-II NCQ 16MB
- **DVD-ROM** LG GSA-H55N bulk black 20x DVD+/-R 10x DL 12x RAM
- **Netzteil** Enermax ELT400AWT 400 Watt Liberty
- **Gehäuse** Sharkoon Rebel 9 Economy Black

Die lokale Festplatte des Hosts, auf der das Betriebssystem, die Virtualisierungssoftware und die Imagedateien mit dem Betriebssystem der Gäste liegen, sind über S-ATA angeschlossen. Wie bereits erwähnt, befinden sich dort nicht die Images, welche für die Festplattentests verwendet werden.

Sowohl das iSCSI-Netzwerk für die Festplattentest, wie auch das dedizierte Netz für die Zeitabfragen sind über eine Gigabit Ethernet mittels eines CAT6 Kabels direkt verbunden, um Leistungsverluste durch Routing oder ähnlichem zu vermeiden.

Das Mainboard unterstützt Dual Channel, was nachfolgend genauer erklärt wird, und die vorhandenen vier identischen Speichermodule sind dementsprechend installiert. Die Taktung

der Module beträgt 800MHz.

Bei der CPU handelt es sich um einen mit 2,4 GHz getakteten Dual-Core Prozessor, der die Befehlssatzerweiterung AMD-V unterstützt, welche im Folgenden genauer beschrieben wird.

#### 3.1.2 Infrastruktur

Zur Konfiguration der Testsoftware, als zuverlässiger Rechner zur Zeitgebung (gewährleistet durch ein natives System), aber auch für die zentrale Speicherung und Verwaltung der Testergebnisse und als Gateway in das öffentliche Netz und zentraler Rechner für den Remotezugriff auf Hosts und Gäste wurde ein Managementrechner eingerichtet. (Abbildung 3.1)

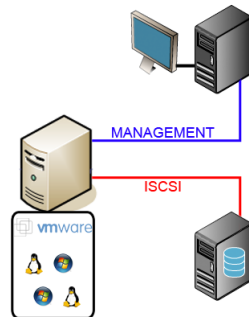


Abbildung 3.1: Hardware Infrastruktur

Auf diesem Managementrechner wurden als dualboot Windows XP und Ubuntu 8.10 (‘‘Intrepid Ibex’’) installiert, da manche Konfigurationstools der Benchmarks ein Windows bzw. ein Linuxsystem benötigen.

Zusätzlich wurde diesem Rechner eine dedizierte Leitung gelegt, über die die Tests des Netzes oder die Zeitabfragen getätigt werden. Über ein extra Netz, dem SAN (‘‘Storage Area Network’’) ist der Hostrechner zusätzlich noch mit dem iSCSI verbunden.

#### Installation der nativen Systeme

Die Installation der einzelnen, insgesamt 4 verschiedenen, nativen Systeme erfolgte auf jeweils separaten Festplatten. Diese Systeme wurden verwendet um unter identischen Voraussetzungen einen Vergleichswert für die VMs zu erhalten. Die Betriebssysteme Windows Server 2003 SP1 RC2 und Ubuntu Linux 9.04 (Jaunty) Server wurden jeweils in den 32- und 64-bit Versionen installiert. Es erfolgte lediglich eine Konfiguration der Netzwerkkarten und sonst keine weiteren Anpassungen und Updates, um so ein identisches Betriebssystem zu erstellen.

#### Installation des Hypervisors und der Virtuellen Maschinen

Die Installation des Hypervisors VMware ESXi 3.5 konnte problemlos und schnell durchgeführt werden. Es war lediglich notwendig die Installationsroutine von der Installation-CD zu booten und die Auswahl der Festplatte zu treffen. Die Partitionierung und die Formatierung sowie die Installation selbst geschah vollautomatisch. Nachher mussten lediglich die Netzwerkkarten konfiguriert werden.

Die Verwaltung der VMs wird nicht auf dem Hostrechner, sondern von einem externen

Rechner vorgenommen. Der Download des Infrastructure Client erfolgt von der integrierten Webserver von VMware, der auf dem Hostrechner läuft und die über die IP des Rechners erreichbar ist. Auf dem externen Rechner, dem Managementrechner, wurde der Infrastructure Client installiert.

### Infrastructure Client

Wie bereits beschrieben wird hier die Installation und Konfiguration der VMs vorgenommen. Beim Erstellen einer neuen VM müssen zuerst ein paar Angaben über die CPU; RAM, Festplatten, etc getroffen werden. Dies erfolgt grafisch über ein Dialogfenster. Danach kann das ISO des Gastbetriebssystem als CD in die VM eingebunden werden und die Installation ganz normal über die grafische Konsole des Infrastructure Clients erfolgen. Wie bereits bei den nativen Systemen wurden auch hier jeweils die 32- und 64-bit Versionen von Windows Server 2003 SP1 RC2 und Ubuntu Linux 9.04 (Jaunty) Server installiert und auch hier nur die Netzwerkkarten konfiguriert. Die Images wurden nachher alle einmal geklont um so die Versionen mit den VMware-Tools zu erstellen. Unter Windows wird hierfür wieder ein Image als CD in die VM eingebunden und kann über einen Setupdialog installiert werden. Unter Linux wird lediglich ein Bashscript ausgeführt, welches die VMware-Tools auch hier problemlos installiert.

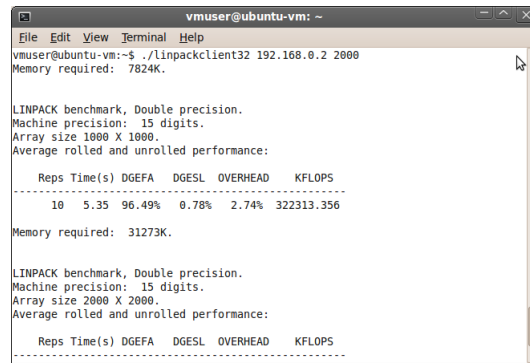
## 3.2 Anpassung der Benchmarktools

Wie bereits in 2.3 beschrieben, müssen die Benchmarktools an die virtuellen Maschinen und deren Zeitgebung angepasst werden. Eventuelle Probleme bei manchen Tools bei der Anpassung an die 32 bzw. 64 bit Versionen der Windows und Linux Systeme werden im Folgenden behandelt. Die hier verwendeten Quelltexte, Patches und Skripte können im Anhang nachgeschlagen werden.

### 3.2.1 CPU

Beim CPU Benchmark fiel die Wahl auf den sehr bekannten Benchmark LINPACK.

LINPACK ist eigentlich eine Programmbibliothek zur Berechnung von linearen Gleichungssystemen. Ursprünglich wurde es in Fortran geschrieben, um damit Supercomputer in den 70er und 80er Jahren zu testen. Mittlerweile ist es jedoch nicht nur eine Programmbibliothek, sondern wird unter Anderem dafür verwendet CPU Leistungen miteinander zu vergleichen, hierfür sind auch verschiedenste Versionen in sehr vielen gängigen Programmiersprachen vorhanden, wie z.B. die hier verwendete Version in C. Der LINPACK-Benchmark misst die Geschwindigkeit der CPU in FLOPS. FLOPS sind floating point operations per second, also die Gleitkommaoperationen pro Sekunde. Als Gleitkommaoperationen bezeichnet man die Befehle und Berechnungen einer CPU unter Verwendung von Gleitkommazahlen. Dieser Wert gibt wider, wie schnell die CPU ein dichtes  $N \times N$  lineares Gleichungssystem mit linearen Gleichungen  $Ax = b$  berechnet. Solche Berechnungen werden häufig in komplexen Systemen verwendet und eignen sich daher gut für eine Geschwindigkeitsmessung. Diese Matrix wird mit der 'Gaussian Elimination with partial pivoting' gelöst. LINPACK bedient sich hierzu der BLAS (Basic Linear Algebra Subprograms) Bibliothek. In unserem Fall haben wir die Systeme jedoch nicht anhand der FLOPS, sondern nur mit der Berechnungszeit verglichen, aus welcher sich die FLOPS problemlos berechnen lassen.



```
vmuser@ubuntu-vm: ~  
File Edit View Terminal Help  
vmuser@ubuntu-vm:~$ ./linpackclient32 192.168.0.2 2000  
Memory required: 7824K.  
  
LINPACK benchmark, Double precision.  
Machine precision: 15 digits.  
Array size 1000 X 1000.  
Average rolled and unrolled performance:  


| Reps | Time(s) | DGEFA  | DGESL | OVERHEAD | KFLOPS     |
|------|---------|--------|-------|----------|------------|
| 10   | 5.35    | 96.49% | 0.78% | 2.74%    | 322313.356 |

  
Memory required: 31273K.  
  
LINPACK benchmark, Double precision.  
Machine precision: 15 digits.  
Array size 2000 X 2000.  
Average rolled and unrolled performance:  


| Reps | Time(s) | DGEFA | DGESL | OVERHEAD | KFLOPS |
|------|---------|-------|-------|----------|--------|
|------|---------|-------|-------|----------|--------|


```

Abbildung 3.2: Ausgabe von LINPACK

## Anpassung

Wie eben erwähnt, ist die Zeitgebung in einer virtuellen Maschine unter Last nicht zwingend korrekt. Da der LINPACK-Benchmark eine sehr hohe Last erzeugt, war hier eine Anpassung des Tools notwendig. Der Standard-LINPACK wurde in eine Server/Client-Architektur umgebaut, um somit eine korrekte Zeitgebung in einer VM über einen Zeit-Server auf einem nativen System zu erreichen. Für Linux wurden die Daten per TCP übertragen, welches jedoch unter Windows manchmal zu Verzögerungen und dadurch Ungenauigkeiten führt. Deshalb wurde für Windows noch eine UDP-Version des Servers erstellt, eine Anpassung der Sockets war für diese Version sowieso notwendig. Auf Client Seite war die Anpassung des LINPACK etwas komplizierter, da der Quelltext selbst, damit eine korrekte Berechnung der FLOPS gewährleistet ist, nur an ein paar Stellen verändert werden darf.

Beim Programmstart wird ein Socket erstellt und eine Verbindung mit dem Server hergestellt. Die für die Zeitgebung implementierte Funktion `second()` wurde um die Übertragung der Zeit über die Sockets erweitert. Desweiteren wurde das Grundgerüst des Benchmarks leicht modifiziert. Die Eingabe der Matrizengröße wurde gelöscht und in einer Schleife die Berechnungen der Größen 1000, 2000, 4000, 8000 festgelegt. Diese Matrizen sollten jeweils 10x berechnet werden. Jedoch sollten nach maximal 20 Minuten die Berechnungen nicht mehr weitergeführt werden, um somit eine sehr lange Dauer der Tests zu vermeiden. Der Test läuft somit komplett selbstständig und identisch durch.

### 3.2.2 RAM

Bei dem Benchmarktool RAMspeed handelt es sich ebenfalls um ein freies Programm. Es steht unter einer eigenen, der GPL ähnlichen Lizenz [RMH]. [Ent].

Obwohl durch den Namen suggeriert, handelt es sich bei RAMspeed nicht um ein reines Benchmarkprogramm um den Hauptspeicher zu testen. Es gibt unterschiedliche Tests, die den Cache der CPU, den Hauptspeicher, aber auch die ALU (“arithmetic logic unit”, deutsch “Arithmetisch-logische Einheit”) und die FPU (“Floating Point Processing Unit”, deutsch “Gleitkommaeinheit”) der CPU testen kann. RAMspeed besteht aus zwei Hauptkomponenten.

- **INTmark** und **FLOATmark**. Mit Hilfe dieser Tests wird die maximal mögliche Cache- und Hauptspeichergeschwindigkeit ermittelt. Dazu werden Daten in der Größe  $2^x$  KB ( $x \in$



```

vmuser@ubuntu-vm:~$ ./ram.sh output.txt
INTmark [writing]
RAMspeed (GENERIC) v2.6.0 by Rhett M. Hollander and Paul V. Bolotoff, 2002-09

8Gb per pass mode

INTEGER & WRITING      1 Kb block: 4869.96 MB/s
INTEGER & WRITING      2 Kb block: 5156.56 MB/s
INTEGER & WRITING      4 Kb block: 5519.51 MB/s
INTEGER & WRITING      8 Kb block: 5558.20 MB/s
INTEGER & WRITING     16 Kb block: 5138.64 MB/s
INTEGER & WRITING     32 Kb block: 5098.94 MB/s
INTEGER & WRITING     64 Kb block: 5526.26 MB/s
INTEGER & WRITING    128 Kb block: 5001.44 MB/s
INTEGER & WRITING    256 Kb block: 5417.70 MB/s
INTEGER & WRITING    512 Kb block: 5321.91 MB/s
INTEGER & WRITING   1024 Kb block: 2686.43 MB/s
INTEGER & WRITING   2048 Kb block: 2044.02 MB/s
INTEGER & WRITING   4096 Kb block: 1948.85 MB/s
INTEGER & WRITING   8192 Kb block: 1870.53 MB/s
INTEGER & WRITING  16384 Kb block: 1953.85 MB/s
INTEGER & WRITING  32768 Kb block: 1914.58 MB/s

```

Abbildung 3.3: Ausgabe von RAMspeed

0, ..., 12), also 1KB bis 4MB, blockweise gelesen bzw. geschrieben. Es wird nicht ständig zwischen Lese- und Schreiboperationen gewechselt, sondern beide unabhängig voneinander, nacheinander durchgeführt. **FLOATmark** spricht dabei die erwähnte FPU an.

- **INTmem** und **FLOATmem**. Diese Tests bestehen jeweils aus vier Untertests mit Namen **Copy**, **Scale**, **Add** und **Triad**. Bei diesen Tests handelt es sich zwar um synthetische Tests, allerdings soll dabei so gut wie möglich ein Realitätsbezug hergestellt werden. Bei **Copy** wird ein Bereich aus dem Hauptspeicher gelesen und an einem anderen Platz geschrieben (mathematisch:  $A = B$ ). Bei **Scale** ebenso, nur wird der Wert vorher mit einer Konstante multipliziert (mathematisch:  $A = m \cdot B$ ). Bei **Add** werden zwei Bereiche gelesen, diese addiert und an einen dritten Bereich gespeichert (mathematisch:  $A = B + C$ ). Der letzte Untertest **Triad** ist eine Kombination aus **Add** und **Scale**, also mathematisch  $A = m \cdot B + C$ . **INTmem** zielt also allein darauf ab, die ALU zu testen. Bei **FLOATmem** handelt es sich dann um eine Kombination aus einem FPU und einem ALU Test.

### Anpassung

Die Zeitmessung in virtuellen Maschinen ist wie unter 2.3 erwähnt vor allem bei kurzen Intervallen ungenau. Ist das Intervall ausreichend lang, sind korrekte Ergebnisse zu erwarten. **RAMspeed** führt im Gegensatz zu **LINPACK** keine hochfrequenten Zeitabfragen durch, sondern misst die Zeit nur vor und nach dem Test. Dadurch war eine Anpassung des Benchmarktools nicht nötig.

Die einzelnen Speichertests die **RAMspeed** unterstützt, können über Parameter angegeben werden. Dabei wird **RAMspeed** eine Zahl mitgegeben, die einen bestimmten Test repräsentiert, z.B.: `./ramspeed -1`, `./ramspeed -2` usw.

Zur einfacheren Durchführung der Tests wurde daher lediglich ein Bash-Script unter Linux, bzw. eine Batch-Datei unter Windows erstellt, mittels der die gewünschten Tests der Reihe nach durchgeführt und in eine Ausgabedatei gepiped wurden. Damit sichergestellt war, dass die Ausgabe auch unmittelbar in die Datei geschrieben wurde und nicht im Hauptspeicher bleibt, wurde danach ein **sync** durchgeführt. Unter Linux ist das Programm **sync** bereits vorhanden, unter Windows kann es von Sysinternals [sysb] heruntergeladen werden [sysa].

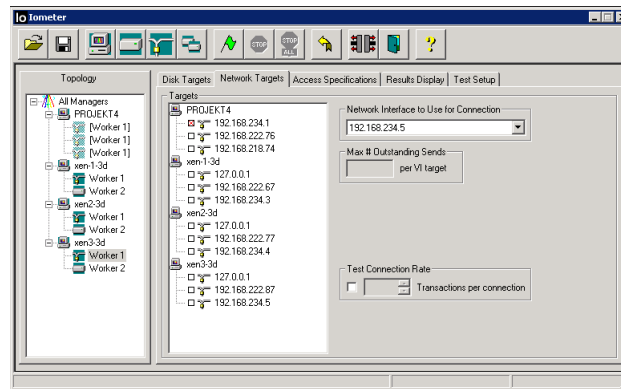


Abbildung 3.4: GUI des Benchmarks Iometer

### 3.2.3 Netz und Disk

Bei den Netzwerk und Festplatten Benchmarks fiel die Wahl auf Iometer, weil dieses Tool die Anforderungen am Besten erfüllte. Iometer ist ein Input/Output Messsystem. Es misst die Performanz unter einer kontrollierten Last und es besteht aus einem Lastgenerator, welcher die I/O Operationen erzeugt, und einem Messwerkzeug. Dieses untersucht die Performanz und zeichnet diese und die Auswirkung auf das System auf. Es kann soweit konfiguriert werden, dass es mehrere verschiedene Szenarien simuliert. Verschiedene Testszenarien mit Festplatten und Netzwerkzugriffen sind hiermit möglich. Auch reine synthetische Benchmarks können durchgeführt werden. Folgende Tests sind denkbar:

- Geschwindigkeitstests der Festplatten- und Netzwerkcontroller
- Bandbreite und Latenztests der Busse .
- Netzwerkdurchsatz der angeschlossenen Karten
- Shared bus Performanz
- Festplatten-Performanz
- Netz Performanz.

Iometer besteht aus zwei Programmen, Iometer und Dynamo. Iometer ist die Kontrolleinheit des Tools. Es besitzt eine graphische Oberfläche mit der man der Last und die Parameter der Tests steuern kann. Iometer teilt einem oder mehreren Dynamos mit, was diese zu tun haben und wertet die gewonnenen Daten aus. Normalerweise läuft immer nur ein Iometer auf einem Server zu dem sich die einzelnen Dynamos verbinden. Der Dynamo generiert die Last. Es besitzt keine Benutzerschnittstelle. Er erledigt und generiert lediglich die Last und erzeugt die Input und Output Operationen für das Netzwerk oder die Festplatten. Es können auch mehrere Dynamos gleichzeitig auf einem System laufen, welche unterschiedliche Lasten erzeugen. Jede laufende Version von Dynamo wird auch als Manager bezeichnet, jeder darin enthaltene Thread als 'worker'.

#### **Anpassung**

Iometer hat bereits eine Server/Client Architektur, weshalb die Zeitgebung in diesem Fall keine Probleme darstellt. Jedoch hatte die verwendete Version einen Bug, welcher gefixed wurde, und für die verschiedenen Systeme, Windows und Linux - jeweils 32 und 64 bit, neu kompiliert werden musste.

Das Problem dabei war, dass nach einem Netzwerkttest Iometer den Endpunkten ein "Stop" mitteilt. Nun beendet aber einer der beiden Endpunkte den Socket und der andere erhält dadurch eine Exception. Dieser Fehler wird von Iometer nicht erkannt und er denkt, dass die Verbindung noch nicht beendet wurde und fällt dadurch in eine Endlosschleife.[pcd]

### *3 Aufbau, Konfiguration und Anpassung*

## 4 Durchführung und Auswertung der Tests

Wie bereits erwähnt, werden zuerst die sogenannten synthetischen Benchmarks mit den bereits angepassten und beschriebenen Tools durchgeführt. Benchmarks ahmen eine besondere Art einer Auslastung auf einem Computer oder einer Komponente nach, um somit Rückschlüsse über die Leistungsfähigkeit zu erhalten. Durch die erhaltenen Ergebnisse wurden die Rahmenbedingungen für die parallelen Benchmarks erstellt und diese dann ausgeführt. Alle Ergebnisse der Tests befinden sich als Tabellen im Anhang.

### 4.1 Synthetische Benchmarks

Bei den synthetischen Benchmarks handelt es sich um spezielle Benchmarks, die sich im Wesentlichen auf eine einzelne Komponente konzentrieren, um damit diese vergleichbar zu machen. In diesem Szenario konnten so die Komponenten in einem nativen System mit einem virtuellen System verglichen werden und ein Wirkungsgrad und der Overhead durch das Verwalten und Scheduling der VM bestimmt werden. Es handelt sich dabei jedoch um Tests, die eine sehr künstliche Situation darstellen. So war es aber möglich, die verschiedenen Systeme, in diesem Fall das native mit einer VM unter VMware, und deren einzelne Hardwarekomponenten untereinander zu vergleichen und den Einfluss der verschiedenen Parameter zu bestimmen.

#### 4.1.1 Durchführung

Bei der Durchführung der Tests konnten alle möglichen Iterationen auf VMware ESXi durchgeführt werden, da es von der technischen Seite keine Einschränkungen gab. Folgende Parameter konnten gewählt werden.

- Betriebssystem (Windows Server 2009 / Ubuntu Linux 9.04)
- Befehlssatzbreite (64 oder 32 bit)
- AMD-Virtualisation (ein / aus)
- VMware-Tools (de- / installiert)

Daraus ergeben sich 16 verschiedene Systeme auf denen die 4 verschiedenen synthetischen Benchmarks durchgeführt und miteinander verglichen wurden. Dies wird nochmal in Abbildung 4.1 verdeutlicht.

#### CPU - Linpack

Um die CPU Tests durchzuführen, wurde der Hostrechner über eine dedizierte Leitung mit dem Managementrechner verbunden. Über diese Leitung ging der Austausch der Zeitgebung

#### 4 Durchführung und Auswertung der Tests

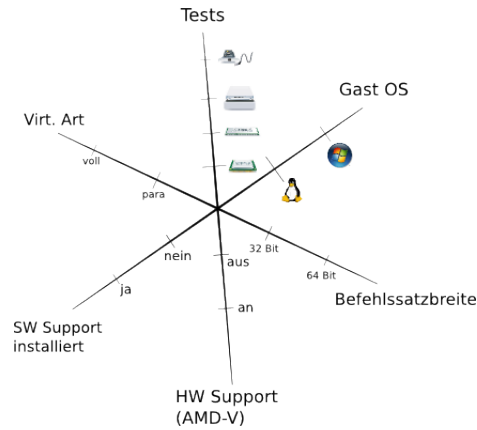


Abbildung 4.1: Mögliche Parameteriteration

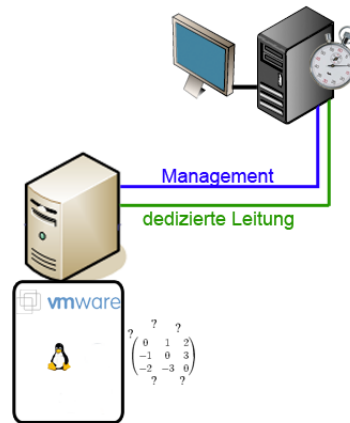


Abbildung 4.2: Aufbau für den CPU Test

für den Test. Auf Serverseite, also dem Managementrechner, wurde der `Linpackserver` gestartet, welcher auf einen Client wartet. Die Befehle zum Starten des Servers unter Linux und Windows sehen also folgendermaßen aus:

```
./linpackserver <port>
```

bzw.

```
linpackserver <hostip> <port>
```

Auf der laufenden virtuellen Maschine, hier auch entweder Linux oder Windows, wurde der Client gestartet, welcher sich über die IP und den Port mit dem Server verbinden konnte.

```
./linpackclient32 <hostip> <port>
```

bzw.

```
./linpackclient64 <hostip> <port>
```

Die Tests liefen nun vollständig und selbstständig, wie bereits in Kapitel 3.2.1 erwähnt, durch und konnten ausgewertet und miteinander verglichen werden.

## RAM - RAMspeed

Wie vorher in Kapitel 3.2.2 schon genauer erläutert, war es hier lediglich notwendig das für `ramspeed` erstellte Skript auf dem zu testenden Rechner zu starten.

```
./ramspeed
```

Danach konnte man die Ausgabedatei auswerten und die verschiedenen Systeme miteinander vergleichen.

## Netz - Iometer

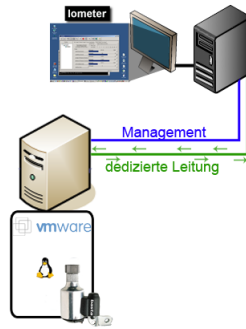


Abbildung 4.3: Aufbau für den Netz Test

IOmeter wird über eine grafische Oberfläche bedient und konfiguriert 3.2.3. Das Programm `iometer.exe`, welches nur für Windows erhältlich ist, ist gleichzeitig externer Zeitgeber und “Ort der Berechnung”. Dieses Programm läuft auf dem Managementrechner, welcher zusätzlich über eine dedizierte Leitung mit dem Hostrechner verbunden ist. Der sogenannte `dynamo`, der die Last erzeugt, wird in der VM gestartet und verbindet sich zur Client-GUI. Auf dem Managementrechner wird automatisch über die GUI ein zweiter `dynamo` gestartet, der den Endpunkt der Netzverbindung darstellt. Als Testnetz wird das dedizierte Netz gewählt, damit die Übertragungen des Benchmarktools selbst den Test nicht beeinflussen. Hierzu werden in der GUI die NICs dieses Netzes als Quelle und Senke gewählt.

Der Aufruf für den `dynamo` ist dabei `./dynamo -m <IP der VM> -i <GUI IP> -n <name>`. Bei der `<GUI IP>` handelt es sich dabei um die IP des Managementrechners, auf der die `iometer.exe` läuft. `<name>` ist ein beliebiger Text zur einfacheren Identifikation des Tests in der Ausgabedatei. Für die Tests wurde ein einheitlicher Ablauf festgelegt und in der GUI eingerichtet. Diese Konfiguration lässt sich als `.icf` Datei abspeichern. Es wurden sowohl Lese- als auch Schreibeoperationen durchgeführt. Jeweils wurden Pakete in den Größen 4, 32 und 256 Kilobyte und 1 und 4 Megabyte verschickt und empfangen. Jede dieser Testgrößen wird drei Minuten getestet und von IOmeter gemittelt (arithmetisches Mittel). Die Lese- und Schreibeoperationen laufen also jeweils 15 Minuten und werden beide in einer `.csv` (“Comma separated value”) Datei gespeichert.

## Disk - Iometer

Für den Festplattentest wird, wie schon erwähnt ein Image das auf einer iSCSI Lun liegt, benötigt. (siehe Kapitel 2.2.4) Dieses Image wird normal wie das normale Festplattenimage der VM eingebunden.

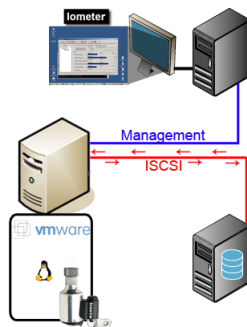


Abbildung 4.4: Aufbau für den Disk Test

Auf der VM wird wie beim Netztest 4.1.1 ein dynamo gestartet, der sich zur iometer.exe verbindet.

Für diesen dynamo wird in der GUI dann der Disktest ausgewählt und als Ziel die ganze Testpartition ausgewählt. Alternativ kann auch auf dem Image ein Dateisystem erstellt werden und dieses als Ziel gewählt werden.

Wie bei dem Netztest werden Lese- und Schreiboperationen durchgeführt. Zusätzlich werden beide Operationen jeweils sequentiell und randomisiert durchgeführt. Die Testgrößen sind identisch zu den Disktests 4, 32, 265 Kilobyte und 1 und 4 Megabyte. Jeder Test läuft auch jeweils drei Minuten, was bei fünf Tests je Einheit eine Stunde Laufzeit ergibt. In der Ausgabedatei werden die Tests nach den vier Einheiten, also Lese- und Schreibtests jeweils sequentiell und randomisiert gruppiert.

#### 4.1.2 Auswertung

Bei der Auswertung der Ergebnisse wurden stets die Windows von den Linux Tests getrennt. Da sie oft, aufgrund verschiedener Implementierungen nur sehr schwer miteinander vergleichbar waren. Hauptsächlich wurde auf den Unterschied zwischen den nativen und virtuellen Systemen eingegangen.

#### CPU - Linpack

Da die verschiedenen Arraygrößen annähernd das gleich Verhalten zeigten, wurde hier Beispielsweise die Arraygröße 2000 gewählt. In Tabelle 4.1 ist schön zu sehen, dass die Leistung der VMs nahezu identisch mit der der nativen Systeme ist.

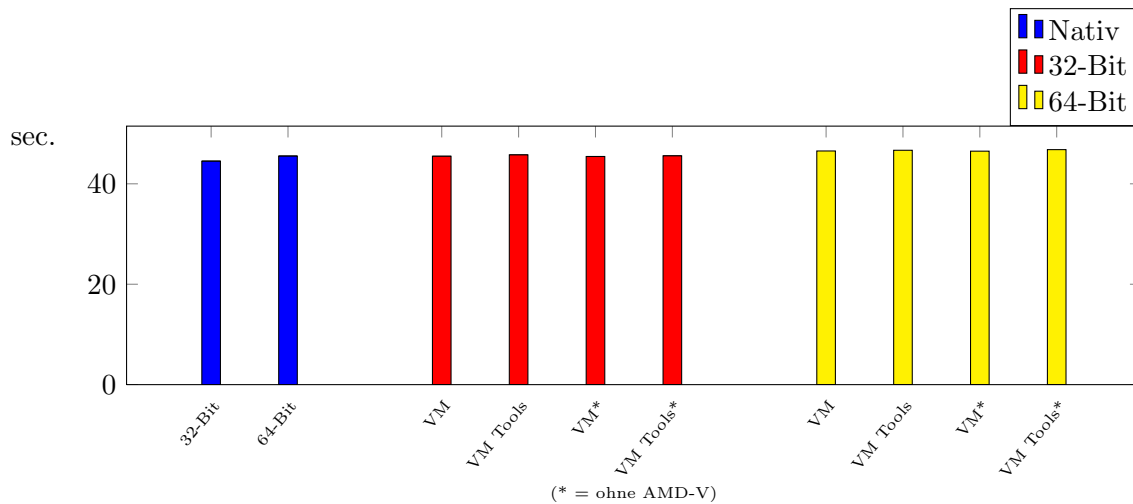
Der Unterschied zwischen den nativen und der virtuellen Maschinen beläuft sich auf weniger als 10%. Damit wird ersichtlich, dass die in 2.2.1 erläuterte Kombination aus "binary translation" und direkter Ausführung der Anweisungen sehr gut funktioniert und nur geringe Verluste, die durch das Scheduling und der Verwaltung der VMs entsteht, mit sich zieht. Bei den Tests mit Windows als Gast-Betriebssystem (Tabelle 4.2) lassen sich nahezu die identischen Werte erzielen und die erzielten Ergebnisse verhalten sich analog.

#### RAM - RAMspeed

RAMspeed testet nicht nur die den Arbeitsspeicher, sondern auch den Level 1 und Level 2 Cache. Da aber, wie schon bei den CPU-Tests, diese Komponente ohne große Verluste



Tabelle 4.1: Linpack Single Linux



durchgereicht werden, ergeben sich nur sehr geringe Unterschiede zwischen den nativen und den virtuellen Systemen. Beispielhaft ist hier deswegen auch nur der Ramtest für Linux in Tabelle 4.3 und für Windows in Tabelle 4.4 aufgeführt.

Zu sehen ist hier, dass die Schreibzugriffe langsamer sind als die Lesezugriffe. Unter Linux ist auch noch insgesamt ein Geschwindigkeitsvorteil von etwa 10% festzustellen, sowohl nativ wie auch virtuell. Die direkten Unterschiede zwischen den virtuellen und den nativen Systemen liegt im Bereich von etwa einem Prozent. Die Technik des virtuellen Arbeitsspeichers mit den “shadow page tables” wie in 2.2.2 genauer erläutert, erfüllt seinen Zweck und die Erwartungen.

### Netz - Iometer

Bei den Iometer Test für das Netz lässt sich die Durchführung nicht einfach auf die physische Komponente weiterreichen, da diese emuliert werden. Bei den Testergebnissen unter Linux, siehe Tabelle 4.5, fällt auf, dass die virtuellen 64-Bit Versionen sehr ähnliche, teilweise sogar minimal bessere Ergebnisse erzielen als das native Vergleichssystem. Dies gelingt ihnen vor allem beim Schreiben, ca.  $49MB/s$  zu ca.  $52,5MB/s$ , also ein um etwa 7% besserer Durchsatz. Bei den 32-Bit Versionen gibt es deutlich größere Unterschiede. Die Aktivierung von AMD-V bringt keine Veränderungen. Jedoch die Installation der vmware-tools. Ohne sie liegt der Durchsatz bei knapp einem Drittel beim Schreiben und beim Lesen sogar noch niedriger bei etwas unter einem Viertel. Mit den vmware-tools bessert sich dies deutlich. Beim Lesen ist fast kein Unterschied festzustellen, beim Schreiben jedoch immer noch ein Verlust von etwa 13%.

Bei den Tests unter Windows erhält man etwas andere Ergebnisse, wie in Tabelle 4.6 gezeigt. Hier ist grundsätzlich bei keinem System ein so deutlicher Einbruch gegeben, jedoch kommt auch keines so nah an die Ergebnisse der nativen Systeme ran. Auch hier sieht man wieder einen Einbruch beim Lesen bei den 32-bit Systemen ohne installierte vmware-tools ( $\approx 12\%$ ).

Sehr auffällig ist aber hier die CPU-Auslastung während der Tests (Tabelle 4.7). So liegt

Tabelle 4.2: Linpack Single Windows

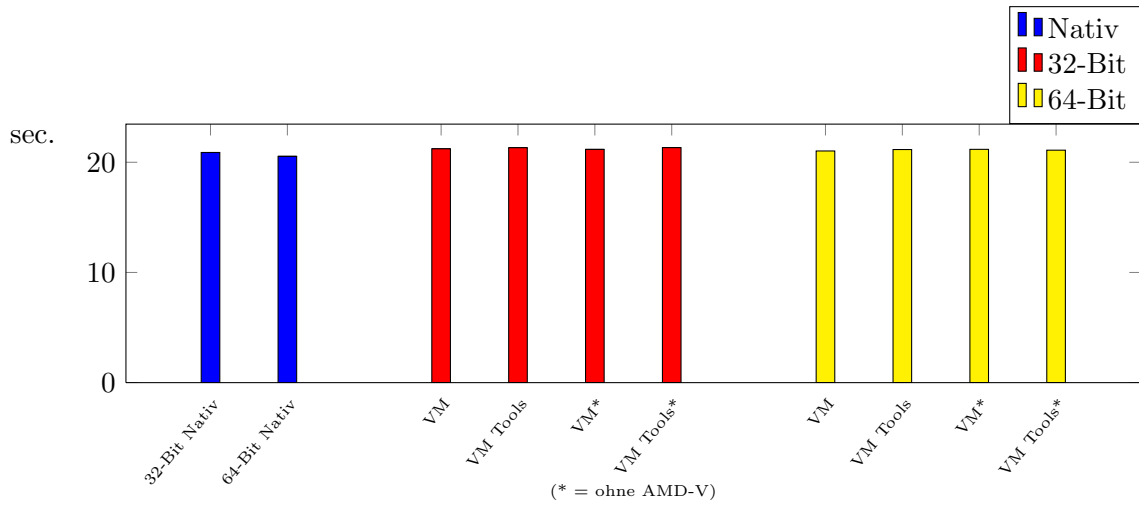
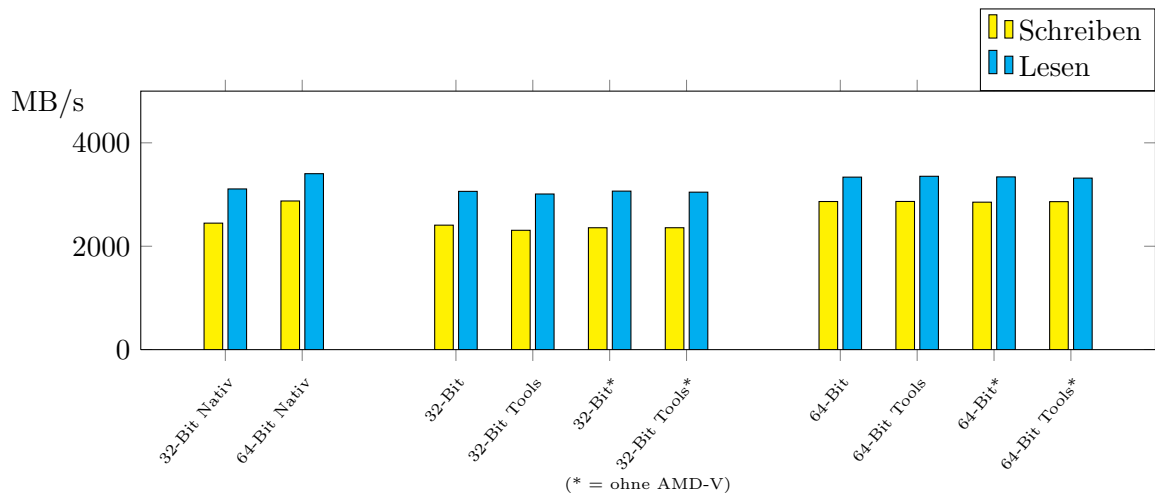


Tabelle 4.3: Hauptspeicher - Linux



diese bei den 32-bit Versionen ohne Einsatz der vmware-tools bei  $\approx 80\%$ . Durch die Installation der vmware-tools wird dieser Wert in etwa halbiert. Bei den 64-bit Versionen ist die Auslastung wesentlich geringer, beim Schreiben bei etwa 10% beim Lesen bei  $\leq 30\%$ .

### Disk - Iometer

Bei den Disk-Tests wurden die Ergebnisse nicht nur in Windows und Linux getrennt, sondern auch in die sequentiellen und die randomisierten Zugriffe. Als erstes werden die sequentiellen Zugriffe unter Linux betrachtet, wie in Tabelle 4.8 zu sehen ist.

Auffällig ist hier, dass der Durchsatz verglichen mit den nativen Systemen schon recht deutlich ( $\approx 25-40\%$ ) einbricht. Das Verhalten der 64-bit und den 32-bit Versionen ist analog.

Tabelle 4.4: Hauptspeicher - Windows

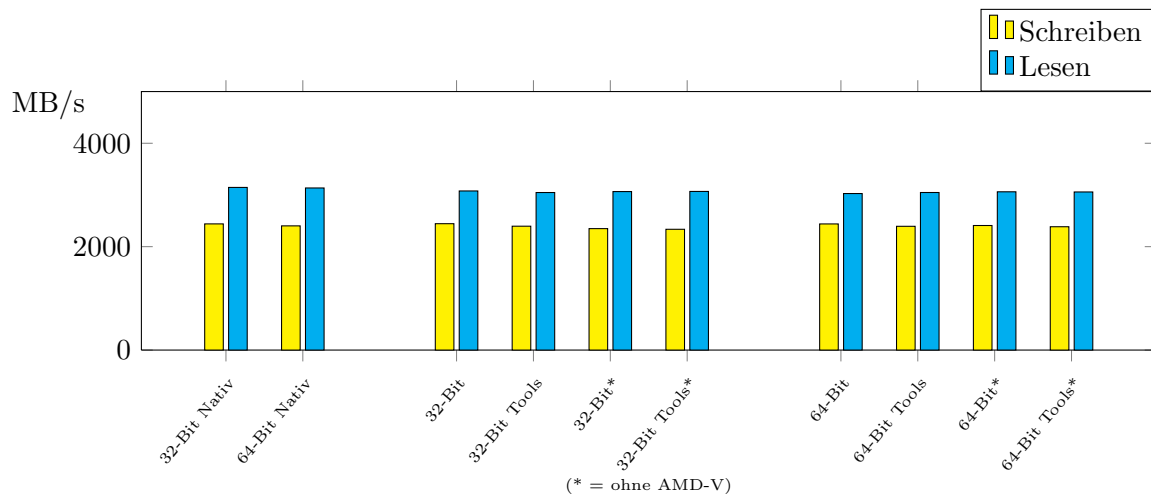
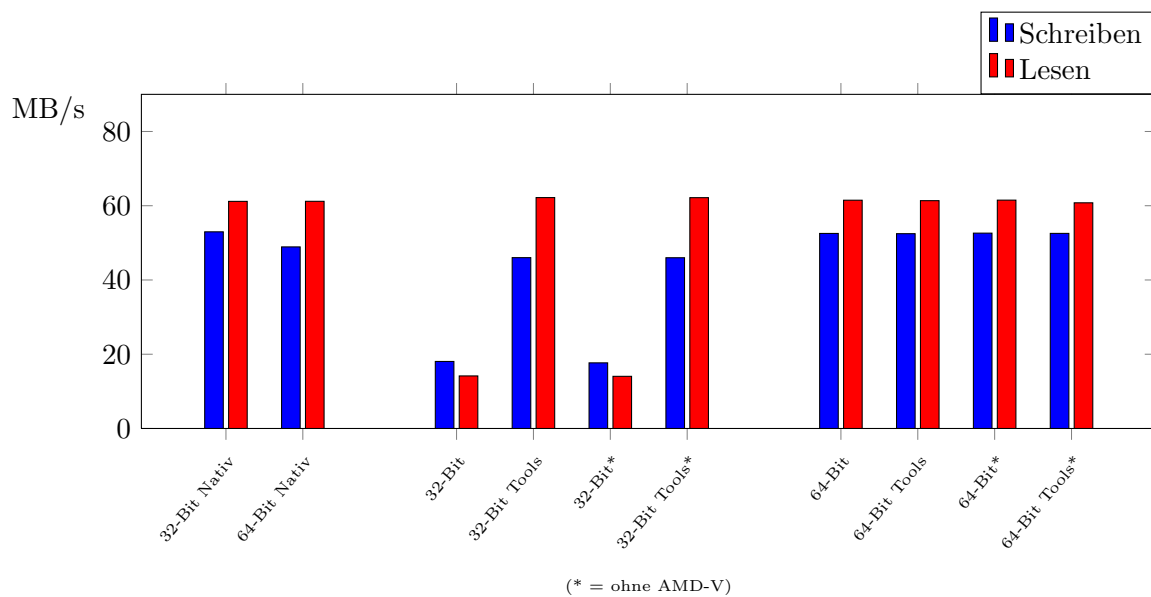


Tabelle 4.5: Net Single Linux



So haben die Versionen ohne aktiviertes AMD-V ein Verminderung von ca. 30-40%. Bei aktiviertem AMD-V fällt auf, dass ohne die vmware-tools das Schreiben ähnlich deutlich einbricht und das Lesen wesentlich geringer im Bereich von  $\approx 10\%$ . Bei den Versionen mit installierten vmware-tools ist der Schreibvorgang schneller als der Lesevorgang. Beim Schreiben ist der Durchsatz sogar geringfügig höher als bei den nativen Systemen. Beim Lesen liegt aber der Verlust bei ca. 20%. Bei den randomisierten Zugriffen sieht es ein wenig anders aus, Tabelle aus 4.9.

Tabelle 4.6: Net Single Windows

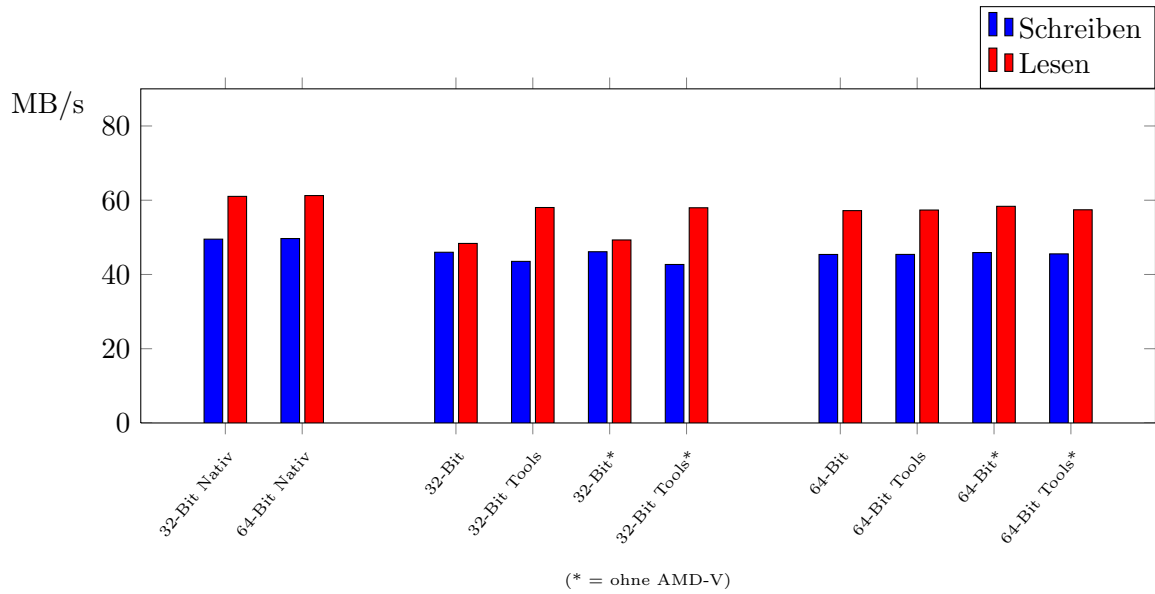
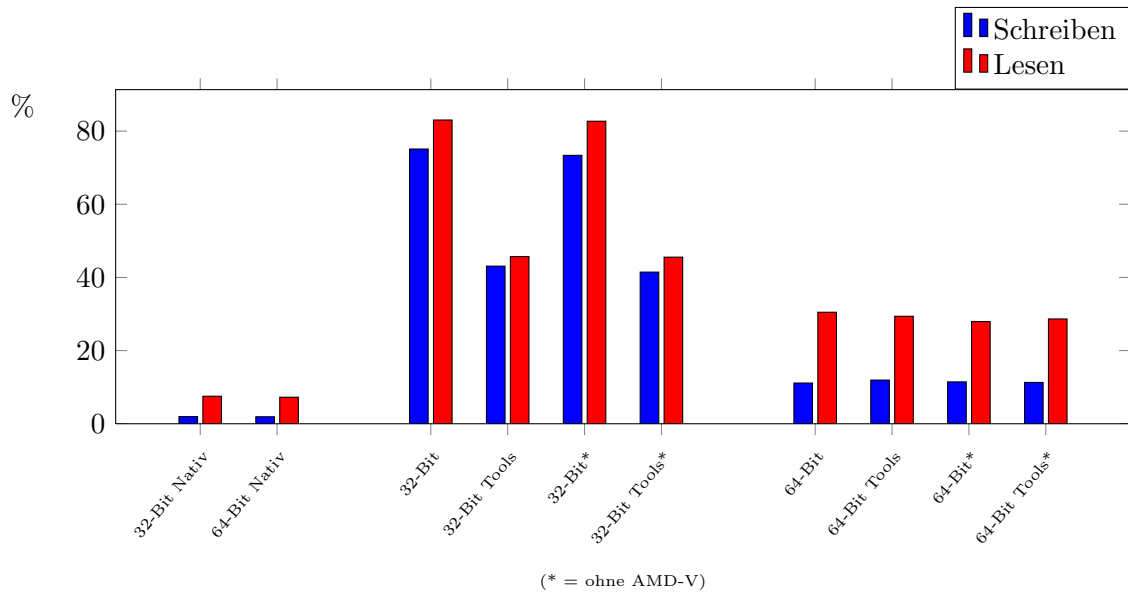


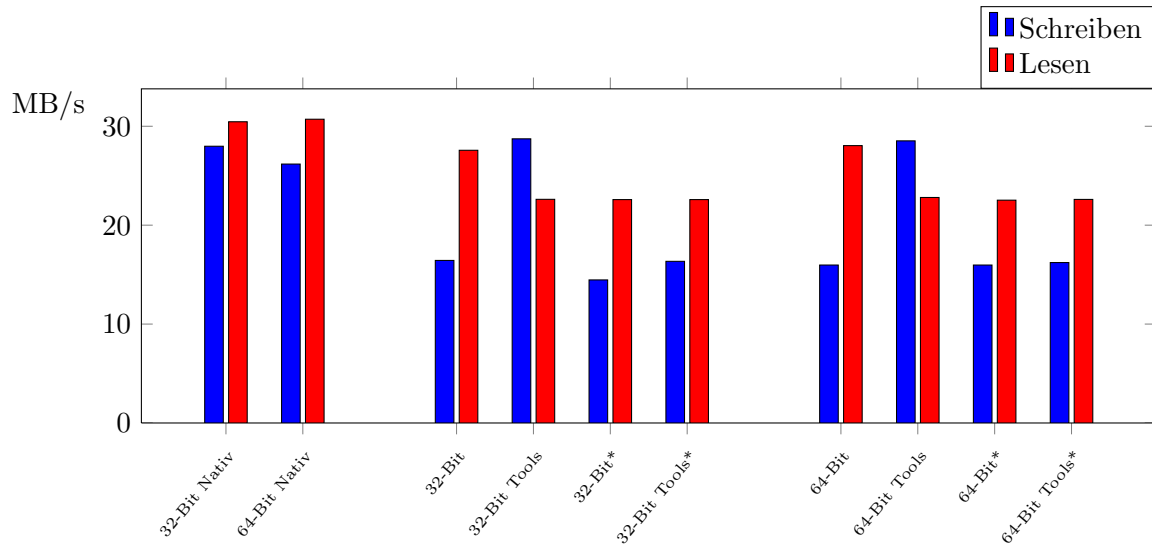
Tabelle 4.7: Net Windows - CPU Auslastung



Hier sind die Verluste beim Lesen durchgehend bei maximal 25%. Desweiteren fällt auf, dass hier die vmware-tools oder auch AMD-V keine besonderen Vor- oder Nachteile hat. Bei den gleichen Tests unter Windows sind die Unterschiede zwischen den Systemen geringer. Als erstes werden auch hier die sequentiellen Zugriffe in Tabelle 4.10 gezeigt.

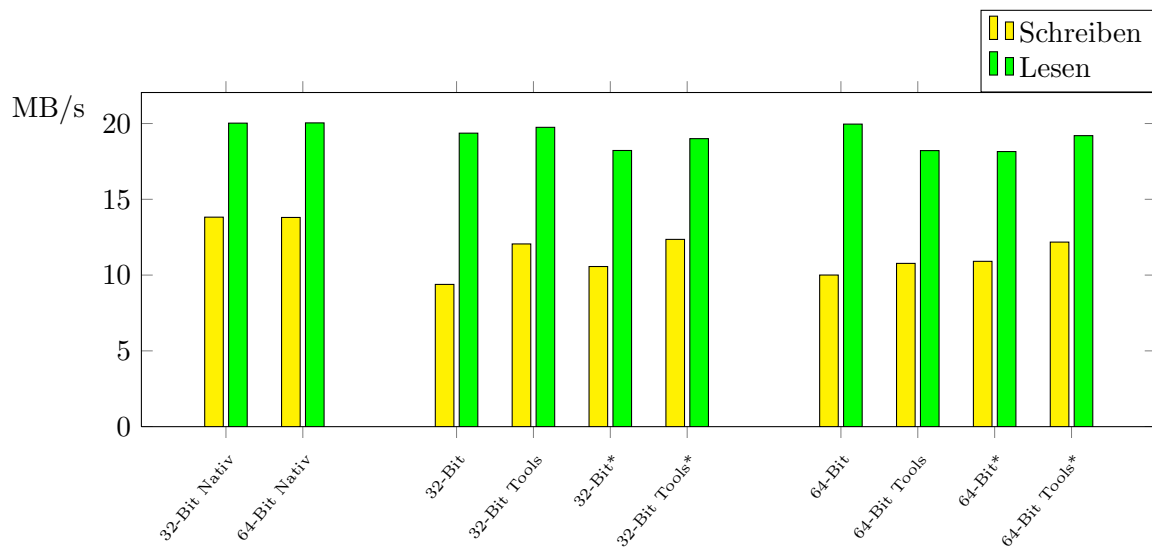
Hier sieht man lediglich einen deutlicheren Verlust bei den virtuellen 32-bit Versionen. Die

Tabelle 4.8: Disk Single Linux sequentiell



(\* = ohne AMD-V)

Tabelle 4.9: Disk Single Linux randomisiert



(\* = ohne AMD-V)

Installation der vmware-tools und die Aktivierung von AMD-V bringt hier kleine Nachteile. Die randomisierten Zugriffe zeigen einige interessante Eigenheiten, siehe Tabelle 4.11.

Tabelle 4.10: Disk Single Windows sequentiell

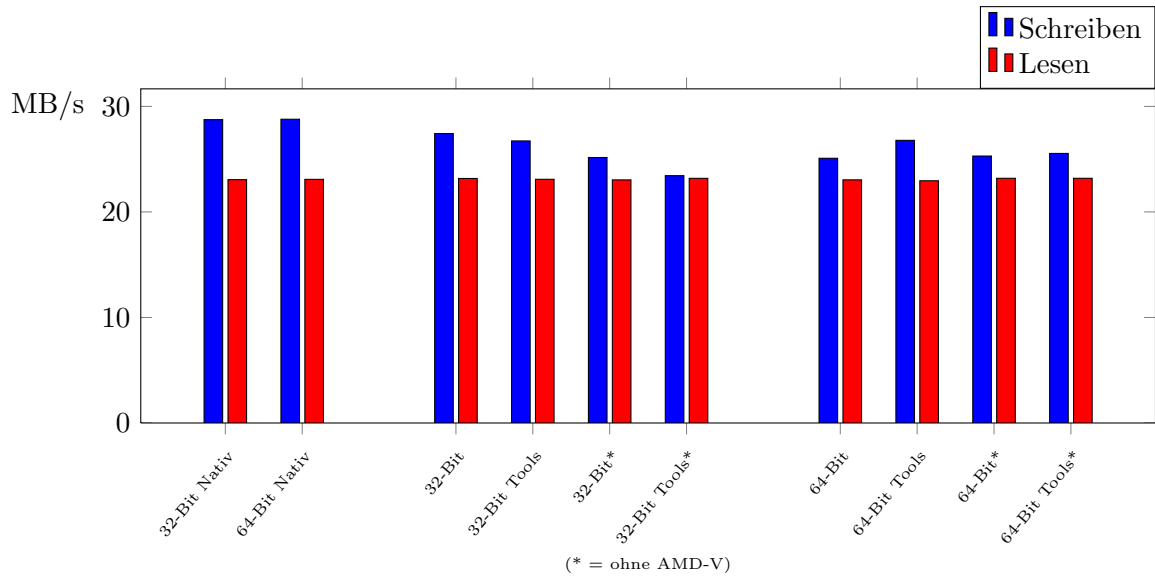
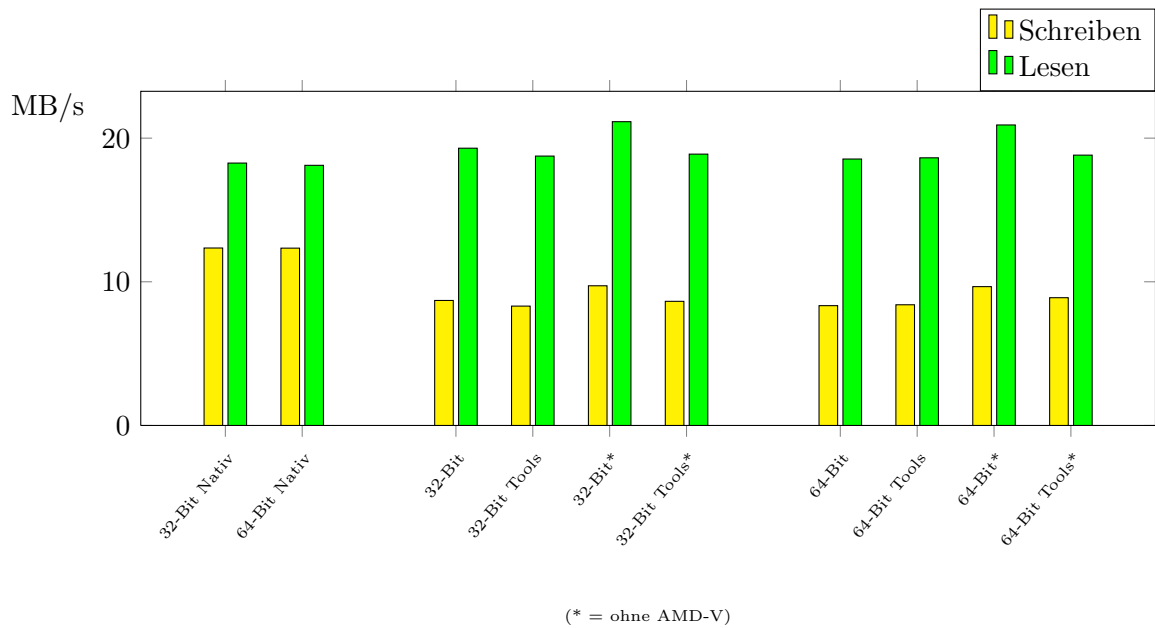


Tabelle 4.11: Disk Single Windows randomisiert



Hier ist auffällig, dass beim Lesen durchgängig ein höherer Durchsatz im Bereich von  $\approx 3\%$  bis  $\approx 13\%$  erreicht werden. Der Einsatz von AMD-V bringt bei 32-bit und 64-bit einen kleinen Vorteil. Der Schreibzugriff erfolgt aber bei den Tests erheblich langsamer als auf den nativen Systemen, bleibt aber immer über 50%.

## 4.2 Parallele Benchmarks

Aufgrund der Ergebnisse aus den synthetischen Tests wurden die parallelen Tests ausgewählt. Jedoch wurden nicht alle Parameteriterationen getestet, sondern nur ein einzige Standardkonfiguration gewählt. WELCHE WAR DAS???

Dabei sollte festgestellt werden, wie skalierfähig VMware ist, was durch den Einsatz mehrerer VMs gleichzeitig und auch eine unterschiedliche oder eine gleiche Auslastung derer bedeutet. So wurde untersucht, wie sich die Leistung verhält, wenn mehrere VMs gleichzeitig dieselbe Komponente beanspruchen. Desweiteren wurde getestet, ob verschiedene Komponenten einen Einfluß aufeinander haben, z.B. ob eine Auslastung der CPU Einbußen bei den Festplatten- oder Netzdurchsatz mit sich zieht.

### 4.2.1 Durchführung

Die erste Reihe der Tests war eine einfache Erweiterung der synthetischen Tests, um die Skalierbarkeit festzustellen. Die bekannten vier Komponenten CPU, RAM, Netz, Festplatte, wurden jeweils mit 2 oder 3 gleichzeitig laufenden VMs auch parallel durchgeführt. Dadurch konnte man feststellen, ob und wieviel Leistung beim Betrieb paralleler VMs verloren oder vielleicht sogar durch geschickten Einsatz von Caching gewonnen werden kann. Die Tests

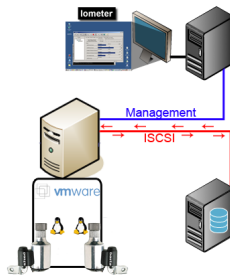


Abbildung 4.5: Aufbau für den Disk Test mit 2 parallel laufenden VMs

wurden mit den gleichen Voraussetzungen wie die synthetischen Tests durchgeführt. Die Kommunikation und Konfiguration erfolgte über das Managementnetzwerk, die Tests selber liefen, wenn erforderlich, über eine dedizierte Leitung und die Zeitgebung erfolgte wie bekannt auf einem externen nativen System. In Abbildung 4.5 ist der Aufbau für den Test mit 2 parallel gestarteten Disktests dargestellt.

Der zweite Teil stellte eine deutlich komplexeres Szenario dar. Zum einen wurden hier die

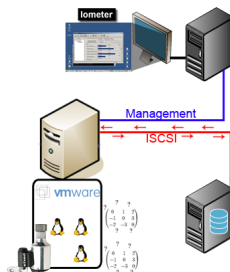


Abbildung 4.6: Aufbau für den Disk Test mit ausgelasteter CPU

synthetischen Disk- und Netztests erweitert. Dazu wurden 2 weitere VMs gestartet, auf denen jeweils ein Linpack-Test lief, um so die CPU auszulasten. In Abbildung 4.6 ist der Aufbau für den Disktest dargestellt. Die Berechnungsdauer des Linpack war nicht interessant, sie diente lediglich dazu, die CPU voll zu belasten, um so eine eventuelle Auswirkung auf die Festplatten- oder Netzzugriffe festzustellen.

Ein weiterer komplexerer Test wird in Abbildung 4.7 dargestellt. Hier wurden 3 VMs gestartet und auf denen jeweils ein Disk- und ein Netztest gestartet. Dadurch sollte untersucht werden, ob die Festplattenzugriffe einen Einfluss auf die Netzzugriffe haben oder andersrum. Der Aufbau der Konfiguration ist wie gewohnt. Die Disktests laufen auf dem iSCSI-Netz

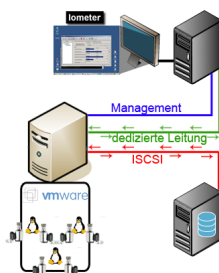


Abbildung 4.7: Aufbau für den Disk- und Netztest mit 3 parallel laufenden VMs

und den Netztests steht eine dedizierte Leitung zur Verfügung. Das Managementnetz dient auch hier nur der Übertragung der Test- und Ergebnisdaten.

Ein weiterer Test war die Netzkommunikation zwischen 2 parallel laufenden VMs. Da hier keine Übertragung auf eine physische Komponente benötigt wird, erwartet man hier einen deutlichen Geschwindigkeitsvorteil.

### 4.2.2 Auswertung

Auch hier werden nicht alle einzelnen Ergebnisse vorgestellt, sondern nur ein paar Eigenheiten und Auffälligkeiten näher erläutert.

#### CPU - Linpack

Bei der Auswertung der CPU-Tests muss berücksichtigt werden, dass es sich beim physischen Rechner um einen Dual-Core-Prozessor handelt und den VMs jeweils eine virtuelle CPU zugewiesen wird. Deshalb kann man erwarten, dass bei 2 parallelen Tests kein oder nur ein sehr geringer Einbruch eintritt. In Tabelle 4.12 zu sehen. Es tritt jedoch ein Verlust von ca. einem Drittel auf, was allein durch das Scheduling von 2 VMs zurückzuführen ist.

Bei 3 parallel gestarteten VMs ergibt sich auch ein, wie in Tabelle 4.13 zu sehen ist, ein noch etwas größeren Nachteil. Hier wurde ein "ideale" Sollwert berechnet, dieser würde eintreten, wenn durch das zusätzliche Verwalten kein zusätzlicher Aufwand benötigt würde. Hier ist auch nur ein relativ kleiner Verlust von ca. 4% festzustellen.

#### RAM - RAMspeed

In den Tabellen 4.14 und 4.15 ist sehr schön zu sehen, dass das Konzept von Dual-Channel 2.2.6 sehr gut funktioniert und auch umgesetzt wird.



Tabelle 4.12: Durchschnitt linpack parallel

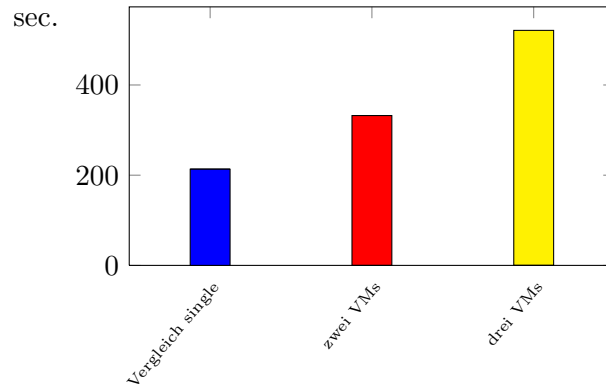


Tabelle 4.13: Durchschnitt linpack parallel - angeglichen

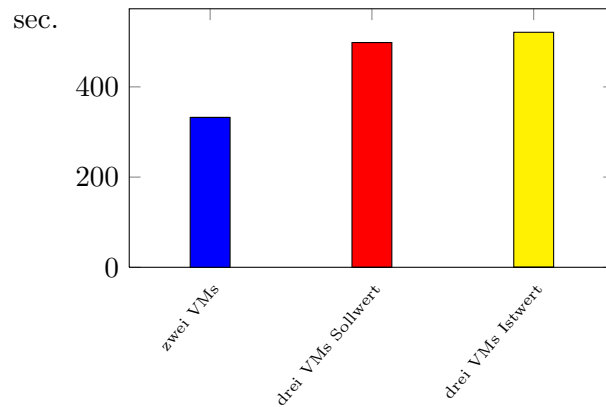
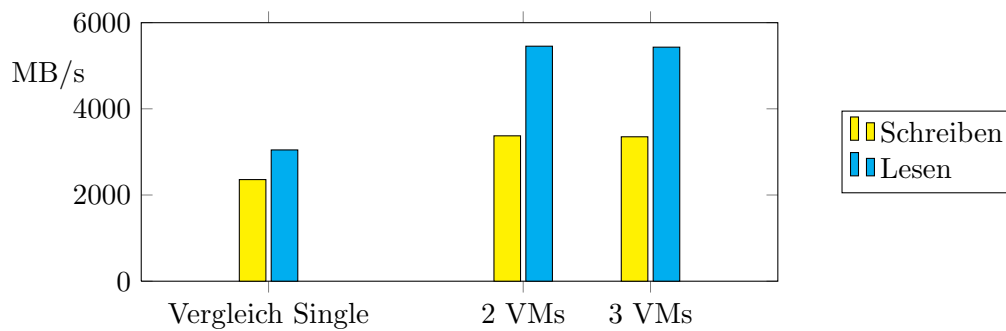
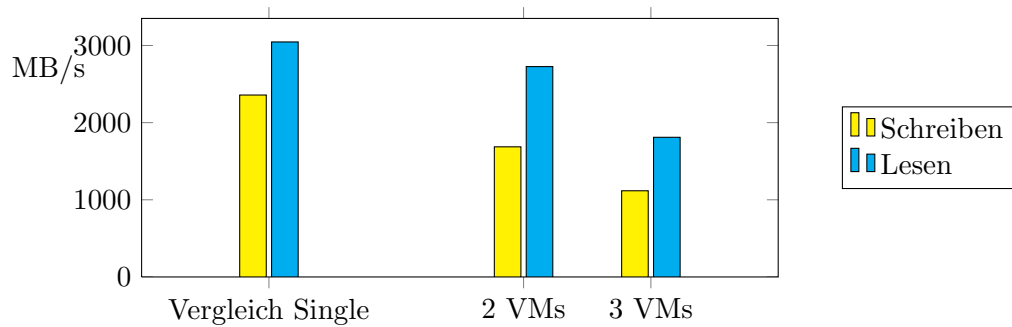


Tabelle 4.14: Hauptspeicher - Summe



Hier ist beim Einsatz von 2 VMs ein Verlust von ca. 10-20% festzustellen. Bei Hinzunahme einer weiteren VM wird sogar insgesamt nahezu der identische Durchsatz erreicht. Der Unterschied zwischen 2 und 3 parallel laufenden VMs ist hier äußerst gering und muss daher nicht genauer betrachtet werden.

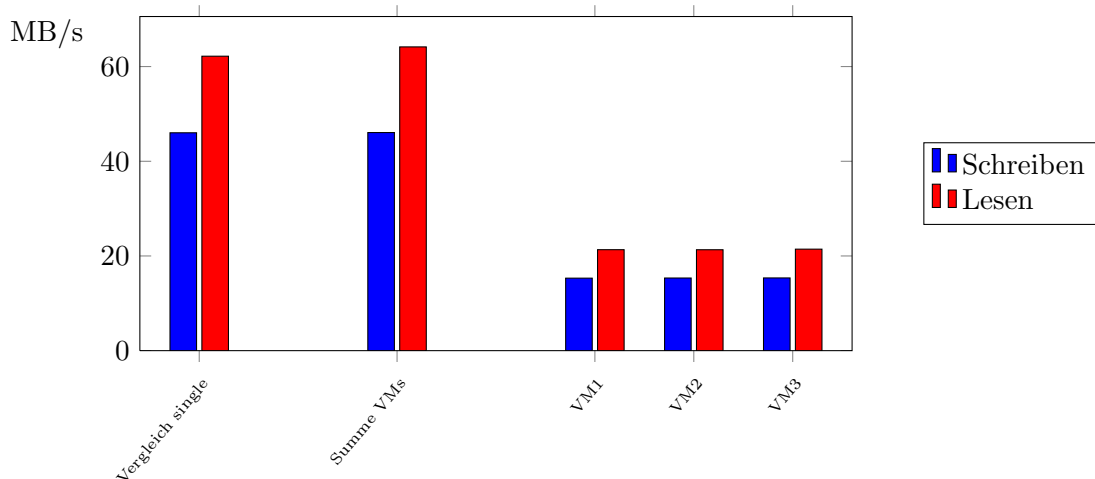
Tabelle 4.15: Hauptspeicher - Durchschnitt



**Netz - Iometer**

Das Verhalten bei 2 oder 3 parallel laufenden Netztests ist analog, aber bei 3 VMs deutlich sichtbar, weshalb hier nur dieser Fall näher betrachtet wird.

Tabelle 4.16: Netz - 3 parallel laufende VMs



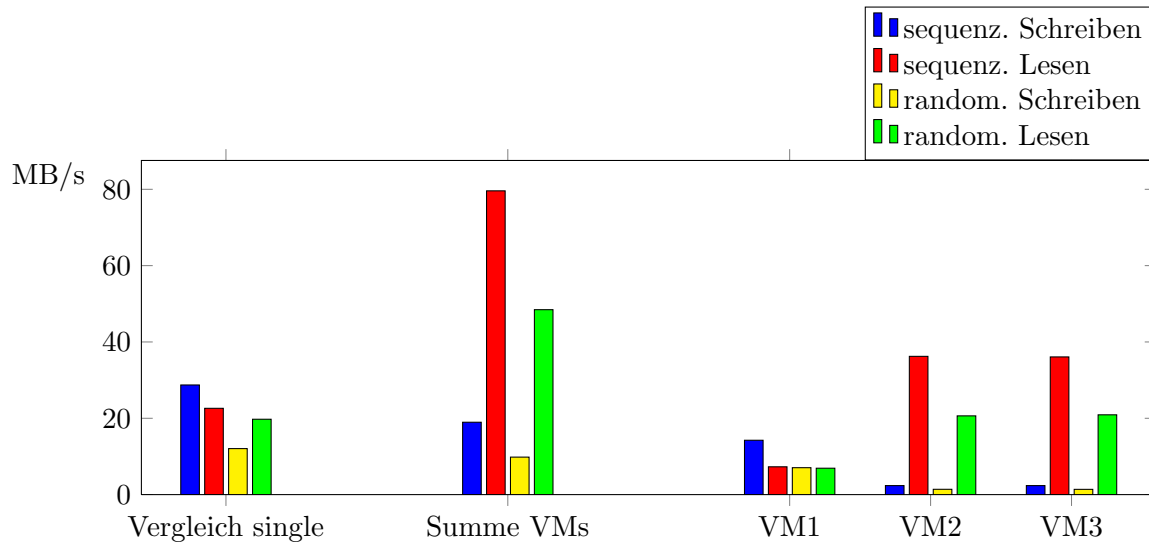
In Tabelle 4.16 ist nichts Auffälliges zu sehen. Die Summe des Durchsatzes beim Lesen und Schreiben entspricht nahezu den Werten aus den synthetischen Tests. Auch die Verteilung auf die parallel laufenden VMs ist nahezu identisch. Hier ist auch der Unterschied sehr gering und kann vernachlässigt werden.

**Disk - Iometer**

Auch hier verhalten sich die Tests mit 2 oder 3 parallel laufenden VMs sehr ähnlich und das Verhalten lässt sich übertragen, weshalb auch hier nur der Test mit den 3 parallel gestarteten Disk-Tests näher betrachtet wird.

In Tabelle 4.17 sieht man sehr deutliche Unterschiede. Zum einen fällt schon einmal beim sequenziellen Lesen bei der Summe der VMs eine deutliche Steigerung auf. Auch beim randomisierten Lesen ist ein deutlicher Zuwachs festzustellen, was darauf hindeutet, dass hier

Tabelle 4.17: Disk - 3 parallel laufende VMs



Caching sehr sinnvoll und effektiv eingesetzt wird. Beim Schreiben ist ein leichter Verlust festzustellen, der sich auf ca. 35% beim sequenziellen und auf ca. 20% beim randomisierten Lesen beläuft. Beim Betrachten des Durchsatzes der einzelnen parallel laufenden VMs fällt hier deutlich auf, dass hier nicht fair und gleichmäßig verteilt wird. Bei VM 2 und VM 3 lässt sich ein ähnliches Verhalten feststellen, jedoch ist dies bei VM 1 deutlich anders. Beim Lesen ist hier ein deutlich niedrigerer Durchsatz als bei den beiden Anderen. Jedoch hat VM 1 beim Schreiben einen deutlichen Vorteil.

#### Disk und Netz unter Last - Iometer

Bei Betrachten der beiden Tabellen 4.18 und 4.19 ist als erstes auffällig, dass bei den Zugriffen auf die Festplatte nahezu keine Nachteile entstehen, während andere parallel laufende VMs eine hohe CPU-Auslastung haben. Es ist nur beim sequenziellen Lesen ein relevanter Verlust von ungefähr 7% festzustellen. Die anderen Werte liegen knapp im Bereich von ca. 2% darunter.

Bei den Netztests unter Last ist der Einbruch des Durchsatzes wesentlich deutlicher. Hier bricht der Lesedurchsatz um fast die Hälfte ein, beim Schreiben entspricht der Verlust fast einem Drittel.

#### Disk und Netz parallel auf 3 VMs - Iometer

Bei den Tests bei mit den 3 gleichzeitig gestarteten VMs darauf jeweils parallel laufenden Disk- und Netztests, fallen mehrere Unregelmäßigkeiten auf. Als erstes wird der Durchsatz der Festplattenzugriffe näher betrachtet, wie in Tabelle 4.20 gezeigt.

Hier fällt auf, dass der gesamte Durchsatz wesentlich geringer ist als im Test aus 4.2.2. Hier ist ein vor allem ein deutlicher Verlust bei den Lesevorgängen zu verzeichnen. Jedoch ist auch die Aufteilung des Durchsatzes, wie schon in 4.2.2 ungleich auf die verschiedenen VMs verteilt.

Tabelle 4.18: Disk unter Last

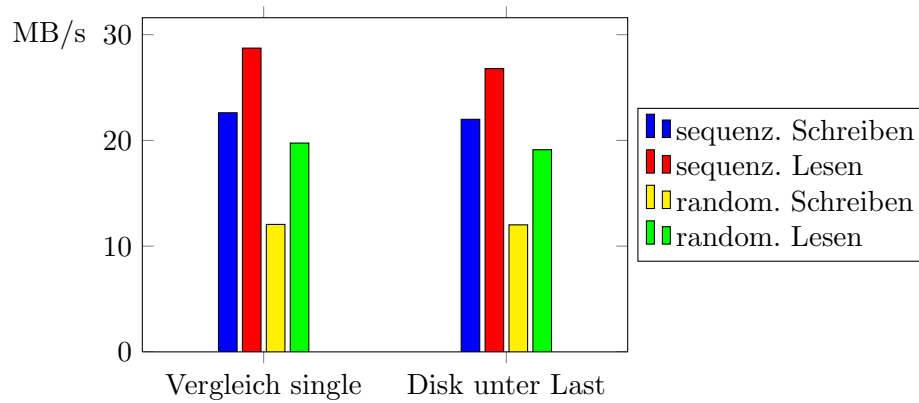
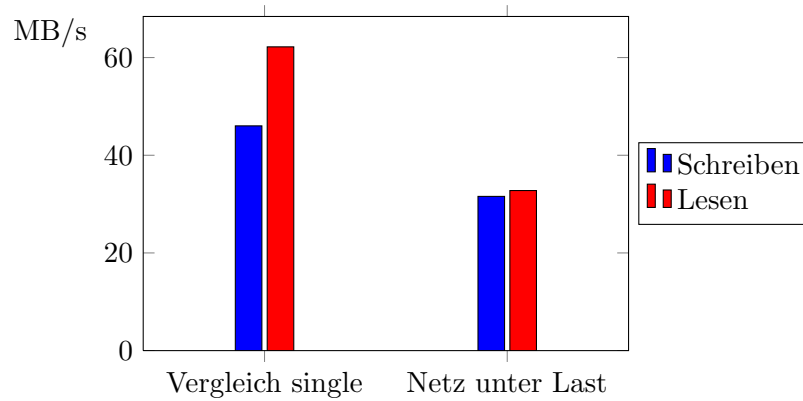


Tabelle 4.19: Netz unter Last



Bei näherer Betrachtung der Ergebnisse der Netztests, siehe Tabelle 4.21, lässt sich jedoch im Vergleich mit dem Test aus 4.2.2 fast kein Verlust feststellen. Hier ist aber die gleichmäßige Aufteilung auf die einzelnen VMs nicht mehr vorhanden. Nur beim Lesen ist dies der Fall, bei den Schreibvorgängen ist der Unterschied zwischen den VMs doch recht deutlich.

### Netzkommunikation zwischen 2 VMs

Da hier keine Übertragung über eine physische Komponente oder physischen Netzes benötigt wird, sollte der Durchsatz deutlich über den normalen Ergebnissen liegen.

Dies ist auch der Fall, wie in Tabelle 4.22 zu sehen ist. Es lässt sich mehr als eine Verdopplung des Durchsatzes feststellen und es ist kein Unterschied mehr zwischen den Lese- und Schreibzugriffen zu sehen.

Tabelle 4.20: Disk aus "Disk und Netz"

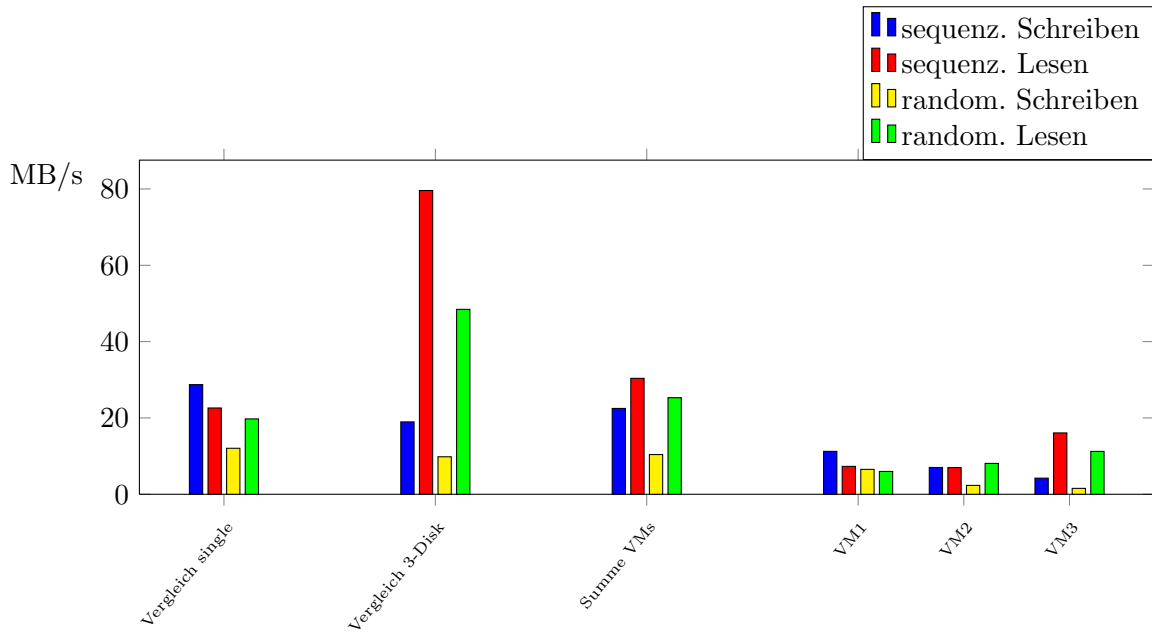


Tabelle 4.21: Netz aus "Disk und Netz"

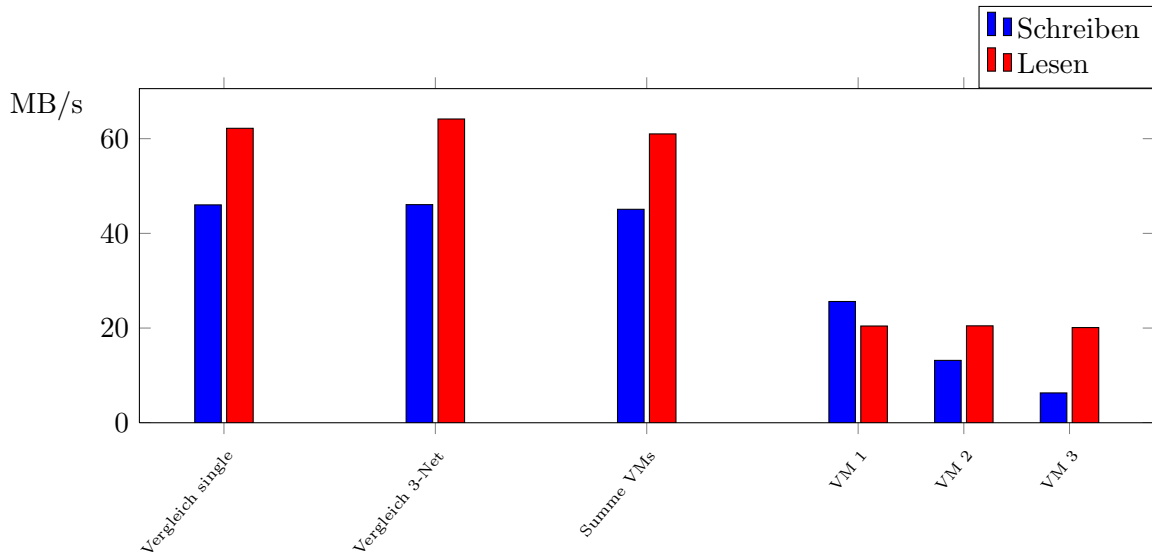
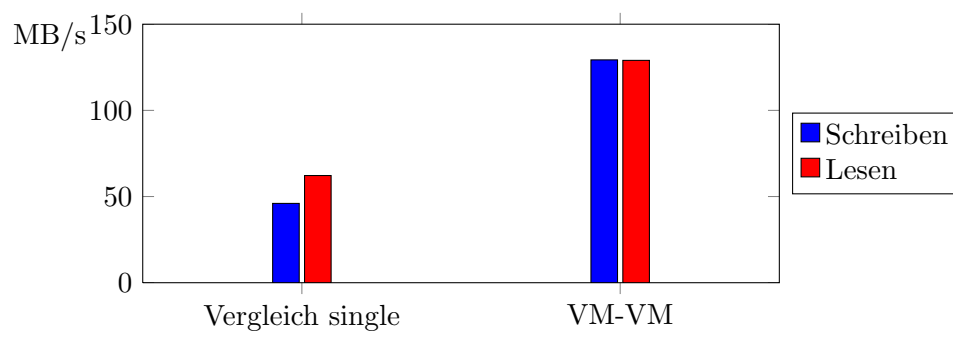


Tabelle 4.22: Netz von VM zu VM



# 5 Fazit und Ausblick

Hier werden noch einmal die wichtigsten Kenntnisse zusammengefasst der Tests zusammengefasst, ein kleines Fazit gezogen und ein Ausblick auf weiterführende Arbeiten gegeben.

## 5.1 Erreichte Ziele

Die Arbeit sollte die Leistungsfähigkeit von VMware ESXi 3.5 unter Verwendung verschiedener Parameter zeigen. Die verwendeten Tests machten die Stärken und die Schwächen des Virtualisierers deutlich. Vor allem im CPU und RAM-Bereich sind fast keine Verluste festzustellen. Durch geschickten Einsatz von Puffern sind teilweise sogar Vorteile bei den Virtualisierungen gegenüber den nativen Systemen zu erkennen. Auch der Verlust und die Verteilung der Gesamtleistung auf mehrere parallel laufende VMs wurde gezeigt. Dadurch kann man einen Eindruck gewinnen, in welchen Einsatzgebieten VMware sinnvoll einsetzbar ist.

## 5.2 Fazit

Bei geschicktem Einsatz des Virtualisierers VMware ESXi lassen sich sehr gute Ergebnisse erzielen. Jedoch sollte man die Ergebnisse nicht alleine betrachten, sondern Vergleiche mit anderen Virtualisierungen machen. Wie bereits erwähnt sind parallel zu dieser Arbeit noch 3 weitere Arbeiten über die Virtualisierer XEN, Hyper-V und Virtuozzo entstanden.

Bei VMware lassen sich in manchen Bereichen nahezu keine Verluste gegenüber nativen Systemen feststellen, bei anderen Einsatzgebieten jedoch relativ große. Hier könnte man noch Optimierungsversuche unternehmen und diese nochmals testen.

## 5.3 Ausblick

Anschließend an die Arbeit könnte man noch weitere Virtualisierungen unter gleichen Voraussetzungen testen und diese dann mit VMware ESXi vergleichen. Es wäre interessant zu testen, ob sich die Ergebnisse auch auf einer anderen Hardware übertragen lässt, ob sich das Verhalten bei nicht nur 3, sondern auch auf 10 oder mehr parallel laufenden VMs genauso darstellt. Hier wäre aber eine deutlich leistungsfähigere Hardware nötig. Desweiteren könnte man aus den hier gewonnenen Kenntnissen noch ein paar Applikationsbenchmarks durchführen, wie etwa den Einsatz verschiedener Server (Datenbank-, Web-, Mail, etc.).





# Abbildungsverzeichnis

2.1	Ringmodell: Übersicht der Virtualisierungsmöglichkeiten . . . . .	5
2.2	Vergleich der beiden Hypervisortypen . . . . .	5
2.3	Virtualisierung des Hauptspeichers . . . . .	7
3.1	Hardware Infrastruktur . . . . .	12
3.2	Ausgabe von LINPACK . . . . .	14
3.3	Ausgabe von RAMspeed . . . . .	15
3.4	GUI des Benchmarks IOmeter . . . . .	16
4.1	Mögliche Parameteriteration . . . . .	20
4.2	Aufbau für den CPU Test . . . . .	20
4.3	Aufbau für den Netz Test . . . . .	21
4.4	Aufbau für den Disk Test . . . . .	22
4.5	Aufbau für den Disk Test mit 2 parallel laufenden VMs . . . . .	29
4.6	Aufbau für den Disk Test mit ausgelasteter CPU . . . . .	29
4.7	Aufbau für den Disk- und Netztest mit 3 parallel laufenden VMs . . . . .	30

## *Abbildungsverzeichnis*

# Tabellenverzeichnis

4.1	Linpack Single Linux . . . . .	23
4.2	Linpack Single Windows . . . . .	24
4.3	Hauptspeicher - Linux . . . . .	24
4.4	Hauptspeicher - Windows . . . . .	25
4.5	Net Single Linux . . . . .	25
4.6	Net Single Windows . . . . .	26
4.7	Net Windows - CPU Auslastung . . . . .	26
4.8	Disk Single Linux sequentiell . . . . .	27
4.9	Disk Single Linux randomisiert . . . . .	27
4.10	Disk Single Windows sequentiell . . . . .	28
4.11	Disk Single Windows randomisiert . . . . .	28
4.12	Durchschnitt linpack parallel . . . . .	31
4.13	Durchschnitt linpack parallel - angeglichen . . . . .	31
4.14	Hauptspeicher - Summe . . . . .	31
4.15	Hauptspeicher - Durchschnitt . . . . .	32
4.16	Netz - 3 parallel laufende VMs . . . . .	32
4.17	Disk - 3 parallel laufende VMs . . . . .	33
4.18	Disk unter Last . . . . .	34
4.19	Netz unter Last . . . . .	34
4.20	Disk aus "Disk und Netz" . . . . .	35
4.21	Netz aus "Disk und Netz" . . . . .	35
4.22	Netz von VM zu VM . . . . .	36



# Ergebnisse

## synthetische Benchmarks

	Nativ	VM	VM Tools	VM*	VM Tools*
CPU - Linpack Windows	in s				
32-bit	44,5	45,49	45,76	45,44	45,57
64-bit	45,51	46,54	46,83	46,51	46,81
CPU - Linpack Linux	in s				
32-Bit	20,89	21,22	21,32	21,17	21,33
64-Bit	20,53	21,02	21,16	21,17	21,1
RAM - RAMspeed Linux	in MB/s				
32-Bit					
Write	2446,68	2407,77	2308,22	2357,58	2357,87
Read	3108035	3061,29	3010,32	3066,14	3045,84
64-Bit					
Write	2875,12	2864,91	2866,06	2852,24	2862,28
Read	3403235	3336,64	3353,08	3341,04	3317,74
L1 Cache - RAMspeed Linux	in MB/s				
32-Bit					
Write	17146,53	16929,1	17031,16	17063,32	17007,32
Read	14554,83	14483,8	14221,06	14386,29	14462,38
64-Bit					
Write	28832,58	28584,96	28579,43	28570,91	28748,93
Read	36831,66	36594,29	35347,76	36624,3	36652,2
L2 Cache - RAMspeed Linux	in MB/s				
32-Bit					
Write	5635,32	6305,2	6330,38	5808,11	6327,65
Read	6608,65	6694,87	7104,37	6654,8	6559,92
64-Bit					
Write	6613,25	7574,16	7314,57	7590,79	7320,25
Read	7261,21	7095,86	7306,44	7757,55	7648,91
RAM - RAMspeed Win	in MB/s				
32-Bit					
Write	2440,52	2444,25	2396	2348,96	2336,24

Ergebnisse

Read	3145,88	3077,44	3047,22	3065,53	3068,26
64-Bit					
Write	2402,26	2439,69	2393,93	2409,24	2385,51
Read	3135,22	3027,55	3047,66	3061,26	3057,97
L1 Cache - RAMspeed Win	in MB/s				
32-Bit					
Write	18829,04	18576,76	18401,03	18464,4	18553,61
Read	19255,9	19102,44	19085,77	18952,56	19171,54
64-Bit					
Write	18721,47	18555,95	18378,34	18570,87	18352,22
Read	19242,95	18941,19	18426,38	19035,1	19017,8
L2 Cache - RAMspeed Win	in MB/s				
32-Bit					
Write	6382,29	6259,77	6260,89	6287,07	6290
Read	7331,69	7202,39	7144,83	7194,56	7200,43
64-Bit					
Write	6387,08	6285,36	6264,11	6283,81	6267,12
Read	7298,16	7140,59	7143,01	7190,53	7146,73
Netz - Iometer Linux	in MB/s				
32-Bit					
Write	52,97	18,05	46,02	17,68	45,99
Read	61,17	14,16	62,19	14,05	62,17
64-Bit					
Write	48,91	52,54	52,47	52,62	52,55
Read	61,19	61,49	61,36	61,51	60,79
Netz - Iometer Windows	in MB/s				
32-Bit					
Write	49,51	45,99	43,53	46,12	42,69
Read	61,04	48,36	58,04	49,3	57,97
64-Bit					
Write	49,68	45,39	45,4	45,91	45,55
Read	61,26	57,2	57,36	58,36	57,43
Netz - Iometer Win CPU	in %				
32-Bit					
Write	1,95	75,11	43,09	73,37	41,46
Read	7,53	83,03	45,7	82,7	45,54
64-Bit					
Write	1,91	11,13	11,94	11,45	11,3
Read	7,26	30,48	29,39	27,94	28,65

Disk - Iometer - Linux	in MB/s				
32-Bit					
Write - sequentiell	27,98	16,44	28,73	14,46	16,34
Read - sequentiell	30,46	27,57	22,61	22,59	22,59
Write - random	13,82	9,38	12,06	10,57	12,36
Read - random	20,03	19,36	19,75	18,22	19
64-Bit					
Write - sequentiell	26,18	15,97	28,52	15,97	16,22
Read - sequentiell	30,71	28,04	22,8	22,53	22,6
Write - random	13,8	10	10,77	10,91	12,18
Read - random	20,04	19,96	18,21	18,15	19,2
Disk - Iometer - Windows	in MB/s				
32-Bit					
Write - sequentiell	28,75	27,43	26,73	25,16	23,44
Read - sequentiell	23,06	23,17	23,09	23,04	23,18
Write - random	12,35	8,7	8,31	9,72	8,64
Read - random	18,26	19,3	18,75	21,15	18,89
64-Bit					
Write - sequentiell	28,79	25,09	26,79	25,3	25,55
Read - sequentiell	23,09	23,04	22,95	23,19	23,19
Write - random	12,34	8,34	8,4	9,66	8,89
Read - random	18,11	18,55	18,63	20,92	18,82

**parallele Benchmarks**

CPU - Linpack parallel	in s		
	single	zwei Vms	zwei Vms
single	213,21		
2 parallel		332,19	
3 parallel			521,07
CPU angepasst	zwei Vms	drei Vms Soll	drei Vms ist
2 parallel	332,19		
3 parallel Soll-Wert		498,29	
3 parallel Ist-Wert			521,07

RAM - RAMspeed	in MB/s					
RAM Average	Single	VM 1	VM 2	VM 1	VM 2	VM 3
Schreiben	2414,73	1784,93	1781,78	1124,97	1127,45	1129,85
Lesen	3122,15	2693,92	2665,68	1655,8	1636,75	1614,7
L2 Average	Single	VM 1	VM 2	VM 1	VM 2	VM 3
Schreiben	5758,99	5973,02	5432,21	3349,9	3231,07	2878,01
Lesen	6637,38	6807,51	6227,65	4027,58	3865,53	3891,77
L1 Average	Single	VM 1	VM 2	VM 1	VM 2	VM 3
Schreiben	17127,32	16253,24	16294,96	13301,14	11685,76	10993,71
Lesen	14538,85	14232,71	13920,8	8555,66	8916,35	9284,77
RAM Summe	Single	2 Vms	3 Vms			
Schreiben	2357,87	3373,22	3350,7			
Lesen	3045,84	5452,74	5430,31			
L2 Summe	Single	2 Vms	3 Vms			
Schreiben	6327,65	10707,99	10888,15			
Lesen	6559,92	13224,91	12846,8			
L1 Summe	Single	2 Vms	3 Vms			
Schreiben	17007,32	33994,03	34209,22			
Lesen	14462,38	28275,5	28996,11			



Netz - Iometer	in MB/s				
InterVM	Single	VM-VM			
Schreiben	46,02	129,29			
Lesen	62,19	129,01			
Netz - Last	Single	VM 1			
Schreiben	46,02	31,58			
Lesen	62,19	32,77			
3-net	Single	Summe Vms	VM1	VM2	VM3
Schreiben	46,016	46,07	15,31	15,34	15,36
Lesen	62,19	64,16	21,33	21,31	21,44
2-net	Single	Summe Vms	VM1	VM2	
Schreiben	46,02	45,8	22,87	22,87	
Lesen	62,19	60,86	30,24	30,55	

Disk - Iometer	in MB/s				
Disk - Last	Single	VM 1			
sequenz, Schreiben	28,73	26,79			
sequenz, Lesen	22,61	22			
random, Schreiben	12,06	12,01			
random, Lesen	19,75	19,12			
3-disk	Single	Summe Vms	VM1	VM2	VM3
sequenz, Schreiben	28,73	18,97	14,24	2,36	2,36
sequenz, Lesen	22,61	79,6	7,29	36,23	36,08
random, Schreiben	12,06	9,83	7,05	1,39	1,38
random, Lesen	19,75	48,46	6,92	20,63	20908
2-disk	Single	Summe Vms	VM1	VM2	
sequenz, Schreiben	28,73	22,22	16,55	5,67	
sequenz, Lesen	22,61	31,25	11,59	19,66	
random, Schreiben	12,06	10,58	7,67	2,91	
random, Lesen	19,75	23,7	9,63	14,07	

*Ergebnisse*

Disk - Netz - Iometer						
3 Netz/Disk (Net)	Single	3-Net	Summe Vms	VM 1	VM 2	VM 3
Schreiben	46,02	46,07	45,09	25,61	13,17	6,3
Lesen	62,19	64,16	61,01	20,43	20,47	20,1
3 Netz/Disk (Disk)	single	Vergleich 3-Disk	Summe Vms	VM1	VM2	VM3
sequenz, Schreiben	28,73	18,97	22,49	11,25	7,02	4,22
sequenz, Lesen	22,61	79,6	30,38	7,31	7	16,07
random, Schreiben	12,06	9,83	10,41	6,53	2,33	1,55
random, Lesen	19,75	48,46	25,31	5,98	8,09	11,24

# Quelltexte und Patches

## Linpack

### Linpack-Server-Linux

Listing 5.1: Linpack-Server-Linux

```
1  /* Linpack Time-Server
2  **
3  ** To compile:  cc -O -o linpackserver linpackserver.c
4  */
5
6  #include <stdio.h>
7  #include <sys/types.h>
8  #include <sys/socket.h>
9  #include <netinet/in.h>
10 #include <time.h>
11 #include <sys/timeb.h>
12 #include <float.h>
13
14 typedef float REAL;
15
16 static struct timeval second  (void);
17
18
19 void error(char *msg)
20 {
21     perror(msg);
22     exit(1);
23 }
24
25 int main(int argc, char *argv[])
26 {
27     int sockfd, newsockfd, portno, clilen;
28     char buffer[256];
29     struct sockaddr_in serv_addr, cli_addr;
30     int n;
31     struct timeval tv1, starttime;
32
33     if (argc < 2) {
34         fprintf(stderr, "ERROR, no port provided\n");
35         exit(1);
36     }
37     sockfd = socket(AF_INET, SOCK_STREAM, 0);
38     if (sockfd < 0)
39         error("ERROR opening socket");
40     bzero((char *) &serv_addr, sizeof(serv_addr));
41     portno = atoi(argv[1]);
42     serv_addr.sin_family = AF_INET;
43     serv_addr.sin_addr.s_addr = INADDR_ANY;
44     serv_addr.sin_port = htons(portno);
45     if (bind(sockfd, (struct sockaddr *) &serv_addr,
46             sizeof(serv_addr)) < 0)
47         error("ERROR on binding");
48     printf("Linpack Time-Server started\n");
49     listen(sockfd,5);
50     clilen = sizeof(cli_addr);
```

```

51     newsockfd = accept(sockfd,
52                       (struct sockaddr *) &cli_addr,
53                       &clilen);
54     if (newsockfd < 0)
55         error("ERROR on accept");
56
57     printf("Client connected\n");
58
59     while (1)
60     {
61         bzero(buffer,256);
62         n = read(newsockfd,buffer,255);
63         if (n < 0) error("ERROR reading from socket");
64
65         if (strcmp(buffer,"start\n") == 0)
66         {
67             bzero(buffer,256);
68             starttime = second();
69             n = write(newsockfd,"start",7);
70             if (n < 0) error("ERROR writing to socket");
71         }
72
73
74
75         if (strcmp(buffer,"quit\n") == 0)
76         {
77             bzero(buffer,256);
78             printf("Goodbye\n");
79             n = write(newsockfd,"quit",6);
80             if (n < 0) error("ERROR writing to socket");
81             bzero(buffer,256);
82             shutdown (sockfd,2);
83             shutdown (newsockfd,2);
84             return 0;
85         }
86
87         if (strcmp(buffer,"time\n") == 0)
88         {
89             bzero(buffer,256);
90             tv1 = second();
91             sprintf(buffer,"%d\n",((tv1.tv_sec - starttime.tv_sec)*1000000)+(tv1.
92                 tv_usec-starttime.tv_usec));
93             n = write(newsockfd,buffer,strlen(buffer));
94             if (n < 0) error("ERROR writing to socket");
95         }
96
97         bzero(buffer,256);
98     }
99     shutdown (sockfd,2);
100    shutdown (newsockfd,2);
101    return 0;
102
103
104    static struct timeval second(void)
105    {
106        struct timeval tv1;
107        gettimeofday(&tv1);
108        return tv1;
109    }

```

## Linpack-Client-Linux

Listing 5.2: Linpack-Client-Linux

```

1  /*
2  **
3  ** To compile: cc -O -o linpackclient linpackclient.c -lm
4  **
5  */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10 #include <time.h>
11 #include <float.h>
12
13 #include <sys/types.h>
14 #include <sys/socket.h>
15 #include <netinet/in.h>
16 #include <netdb.h>
17
18
19
20 #define DP
21
22 #ifdef SP
23 #define ZERO          0.0
24 #define ONE           1.0
25 #define PREC         "Single"
26 #define BASE10DIG    FLT_DIG
27
28 typedef float      REAL;
29 #endif
30
31 #ifdef DP
32 #define ZERO          0.0e0
33 #define ONE           1.0e0
34 #define PREC         "Double"
35 #define BASE10DIG    DBL_DIG
36
37 typedef double     REAL;
38 #endif
39
40 static REAL linpack  (long nreps,int arsize);
41 static void matgen   (REAL *a,int lda,int n,REAL *b,REAL *norma);
42 static void dgefa    (REAL *a,int lda,int n,int *ipvt,int *info,int roll);
43 static void dgesl    (REAL *a,int lda,int n,int *ipvt,REAL *b,int job,int roll);
44 static void daxpy_r  (int n,REAL da,REAL *dx,int incx,REAL *dy,int incy);
45 static REAL ddot_r   (int n,REAL *dx,int incx,REAL *dy,int incy);
46 static void dscal_r  (int n,REAL da,REAL *dx,int incx);
47 static void daxpy_ur (int n,REAL da,REAL *dx,int incx,REAL *dy,int incy);
48 static REAL ddot_ur  (int n,REAL *dx,int incx,REAL *dy,int incy);
49 static void dscal_ur (int n,REAL da,REAL *dx,int incx);
50 static int idamax    (int n,REAL *dx,int incx);
51 static REAL second   (void);
52
53 static void *mempool;
54
55 // socket
56 int sockfd, portno, n;
57 struct sockaddr_in serv_addr;
58 struct hostent *server;
59 char buffer[256];
60
61
62
63 void main(int argc, char *argv[])
64 {
65

```

```

66     char    buf[80];
67     int     arsize, i;
68     long    arsize2d, memreq, nreps;
69     size_t  malloc_arg;
70
71     // Create socket and connect to server
72     if (argc < 3)
73     {
74         fprintf(stderr, "usage %s hostname port\n", argv[0]);
75         exit(0);
76     }
77     portno = atoi(argv[2]);
78     sockfd = socket(AF_INET, SOCK_STREAM, 0);
79     if (sockfd < 0)
80         error("ERROR opening socket");
81     server = gethostbyname(argv[1]);
82     if (server == NULL)
83     {
84         fprintf(stderr, "ERROR, no such host\n");
85         exit(0);
86     }
87     bzero((char *) &serv_addr, sizeof(serv_addr));
88     serv_addr.sin_family = AF_INET;
89     bcopy((char *)server->h_addr,
90         (char *)&serv_addr.sin_addr.s_addr,
91         server->h_length);
92     serv_addr.sin_port = htons(portno);
93     if (connect(sockfd, &serv_addr, sizeof(serv_addr)) < 0)
94         error("ERROR connecting");
95
96
97     bzero(buffer, 256);
98     sprintf(buffer, "start\n");
99     n = write(sockfd, buffer, strlen(buffer));
100    if (n < 0)
101        error("ERROR writing to socket");
102    bzero(buffer, 256);
103    n = read(sockfd, buffer, 255);
104    if (n < 0)
105        error("ERROR reading from socket");
106    if (strcmp(buffer, "start") != 0)
107    {
108        error("ERROR from server");
109    }
110
111    for (i=1000; i<8001; i*=2)
112    {
113        //printf("Enter array size (q to quit) [200]: ");
114        //fgets(buf, 79, stdin);
115        //if (buf[0]=='q' || buf[0]=='Q')
116            // break;
117        //if (buf[0]=='\0' || buf[0]=='\n')
118            // arsize=200;
119        //else
120            // arsize=atoi(buf);
121        arsize = i;
122        arsize/=2;
123        arsize*=2;
124        if (arsize<10)
125        {
126            printf("Too small.\n");
127            continue;
128        }
129        arsize2d = (long)arsize*(long)arsize;
130        memreq=arsize2d*sizeof(REAL)+(long)arsize*sizeof(REAL)+(long)arsize*sizeof(
            int);

```

```

131     printf("Memory required:  %ldK.\n", (memreq+512L)>>10);
132     malloc_arg=(size_t)memreq;
133     if (malloc_arg!=memreq || (mempool=malloc(malloc_arg))==NULL)
134     {
135         printf("Not enough memory available for given array size.\n\n");
136         continue;
137     }
138     printf("\n\nLINPACK benchmark,  %s precision.\n",PREC);
139     printf("Machine precision:  %d digits.\n",BASE10DIG);
140     printf("Array size %d X %d.\n",arsize,arsize);
141     printf("Average rolled and unrolled performance:\n\n");
142     printf("      Reps Time(s) DGEFA  DGESL  OVERHEAD   KFLOPS\n");
143     printf("-----\n");
144     nreps=1;
145
146     //      bzero(buffer,256);
147     //      sprintf(buffer,"start\n");
148     //      n = write(sockfd,buffer,strlen(buffer));
149     //      if (n < 0)
150     //          error("ERROR writing to socket");
151     //      bzero(buffer,256);
152     //      n = read(sockfd,buffer,255);
153     //      if (n < 0)
154     //          error("ERROR reading from socket");
155     //      if (strcmp(buffer,"start") != 0)
156     //      {
157     //          error("ERROR from server");
158     //      }
159
160     linpack(10,arsize);
161     free(mempool);
162     printf("\n");
163 }
164
165     bzero(buffer,256);
166     sprintf(buffer,"quit\n");
167     n = write(sockfd,buffer,strlen(buffer));
168     if (n < 0)
169         error("ERROR writing to socket");
170     bzero(buffer,256);
171     n = read(sockfd,buffer,255);
172     if (n < 0)
173         error("ERROR reading from socket");
174     if (strcmp(buffer,"quit") == 0)
175     {
176         printf("Quitting linpack\n");
177     }
178     shutdown(sockfd,2);
179     return 0;
180 }
181
182
183 static REAL linpack(long nreps,int arsize)
184
185 {
186     REAL  *a,*b;
187     REAL  norma,t1,kflops,tdgesl,tdgefa,totalt,toverhead,ops;
188     int   *ipvt,n,info,lda,realnreps;
189     long  i,arsize2d;
190
191     lda = arsize;
192     n = arsize/2;
193     arsize2d = (long)arsize*(long)arsize;
194     ops=((2.0*n*n*n)/3.0+2.0*n*n);
195     a=(REAL *)mempool;
196     b=a+arsize2d;

```

```

197     ipvt=(int *)&b[arsize];
198     realnreps = 0;
199     tdgesl=0;
200     tdgefa=0;
201     totalt=second();
202     for (i=0;i<nreps;i++)
203     {
204         if (second() < 1800) {
205             realnreps++;
206             matgen(a,lda,n,b,&norma);
207             t1 = second();
208             dgefa(a,lda,n,ipvt,&info,1);
209             tdgefa += second()-t1;
210             t1 = second();
211             dgesl(a,lda,n,ipvt,b,0,1);
212             tdgesl += second()-t1;
213         } else break;
214     }
215     for (i=0;i<realnreps;i++)
216     {
217         matgen(a,lda,n,b,&norma);
218         t1 = second();
219         dgefa(a,lda,n,ipvt,&info,0);
220         tdgefa += second()-t1;
221         t1 = second();
222         dgesl(a,lda,n,ipvt,b,0,0);
223         tdgesl += second()-t1;
224     }
225     totalt=second()-totalt;
226     if (totalt<0.5 || tdgefa+tdgesl<0.2)
227         return(0.);
228     kflops=2.*realnreps*ops/(1000.*(tdgefa+tdgesl));
229     toverhead=totalt-tdgefa-tdgesl;
230     if (tdgefa<0.)
231         tdgefa=0.;
232     if (tdgesl<0.)
233         tdgesl=0.;
234     if (toverhead<0.)
235         toverhead=0.;
236     printf("%8ld %6.2f %6.2f%% %6.2f%% %6.2f%% %9.3f\n",
237           realnreps,totalt,100.*tdgefa/totalt,
238           100.*tdgesl/totalt,100.*toverhead/totalt,
239           kflops);
240     return(totalt);
241 }
242
243
244 /*
245 ** For matgen,
246 ** We would like to declare a[][lda], but c does not allow it. In this
247 ** function, references to a[i][j] are written a[lda*i+j].
248 **/
249 static void matgen(REAL *a,int lda,int n,REAL *b,REAL *norma)
250
251 {
252     int init,i,j;
253
254     init = 1325;
255     *norma = 0.0;
256     for (j = 0; j < n; j++)
257         for (i = 0; i < n; i++)
258             {
259                 init = (int)((long)3125*(long)init % 65536L);
260                 a[lda*j+i] = (init - 32768.0)/16384.0;
261                 *norma = (a[lda*j+i] > *norma) ? a[lda*j+i] : *norma;
262             }

```



```

263     for (i = 0; i < n; i++)
264         b[i] = 0.0;
265     for (j = 0; j < n; j++)
266         for (i = 0; i < n; i++)
267             b[i] = b[i] + a[lda*j+i];
268     }
269
270
271     /*
272     **
273     ** DGEFA benchmark
274     **
275     ** We would like to declare a[][lda], but c does not allow it. In this
276     ** function, references to a[i][j] are written a[lda*i+j].
277     **
278     ** dgefa factors a double precision matrix by gaussian elimination.
279     **
280     ** dgefa is usually called by dgeco, but it can be called
281     ** directly with a saving in time if rcond is not needed.
282     ** (time for dgeco) = (1 + 9/n)*(time for dgefa) .
283     **
284     ** on entry
285     **
286     **     a       REAL precision[n][lda]
287     **             the matrix to be factored.
288     **
289     **     lda     integer
290     **             the leading dimension of the array a .
291     **
292     **     n       integer
293     **             the order of the matrix a .
294     **
295     ** on return
296     **
297     **     a       an upper triangular matrix and the multipliers
298     **             which were used to obtain it.
299     **             the factorization can be written a = l*u where
300     **             l is a product of permutation and unit lower
301     **             triangular matrices and u is upper triangular.
302     **
303     **     ipvt    integer[n]
304     **             an integer vector of pivot indices.
305     **
306     **     info    integer
307     **             = 0 normal value.
308     **             = k if u[k][k].eq. 0.0 . this is not an error
309     **             condition for this subroutine, but it does
310     **             indicate that dgesl or dgedi will divide by zero
311     **             if called. use rcond in dgeco for a reliable
312     **             indication of singularity.
313     **
314     ** linpack. this version dated 08/14/78 .
315     ** cleve moler, university of New Mexico, argonne national lab.
316     **
317     ** functions
318     **
319     ** blas daxpy,dscal,idamax
320     **
321     */
322     static void dgefa(REAL *a,int lda,int n,int *ipvt,int *info,int roll)
323
324     {
325     REAL t;
326     int idamax(),j,k,kp1,l,nm1;
327
328     /* gaussian elimination with partial pivoting */

```

```

329
330     if (roll)
331     {
332         *info = 0;
333         nm1 = n - 1;
334         if (nm1 >= 0)
335             for (k = 0; k < nm1; k++)
336             {
337                 kp1 = k + 1;
338
339                 /* find l = pivot index */
340
341                 l = idamax(n-k,&a[lda*k+k],1) + k;
342                 ipvt[k] = l;
343
344                 /* zero pivot implies this column already
345                  triangularized */
346
347                 if (a[lda*k+1] != ZERO)
348                 {
349
350                     /* interchange if necessary */
351
352                     if (l != k)
353                     {
354                         t = a[lda*k+1];
355                         a[lda*k+1] = a[lda*k+k];
356                         a[lda*k+k] = t;
357                     }
358
359                     /* compute multipliers */
360
361                     t = -ONE/a[lda*k+k];
362                     dscal_r(n-(k+1),t,&a[lda*k+k+1],1);
363
364                     /* row elimination with column indexing */
365
366                     for (j = kp1; j < n; j++)
367                     {
368                         t = a[lda*j+1];
369                         if (l != k)
370                         {
371                             a[lda*j+1] = a[lda*j+k];
372                             a[lda*j+k] = t;
373                         }
374                         daxpy_r(n-(k+1),t,&a[lda*k+k+1],1,&a[lda*j+k+1],1);
375                     }
376                 }
377                 else
378                     (*info) = k;
379             }
380         ipvt[n-1] = n-1;
381         if (a[lda*(n-1)+(n-1)] == ZERO)
382             (*info) = n-1;
383     }
384     else
385     {
386         *info = 0;
387         nm1 = n - 1;
388         if (nm1 >= 0)
389             for (k = 0; k < nm1; k++)
390             {
391                 kp1 = k + 1;
392
393                 /* find l = pivot index */
394

```

```

395         l = idamax(n-k,&a[lda*k+k],1) + k;
396         ipvt[k] = l;
397
398         /* zero pivot implies this column already
399         triangularized */
400
401         if (a[lda*k+1] != ZERO)
402         {
403
404             /* interchange if necessary */
405
406             if (l != k)
407             {
408                 t = a[lda*k+1];
409                 a[lda*k+1] = a[lda*k+k];
410                 a[lda*k+k] = t;
411             }
412
413             /* compute multipliers */
414
415             t = -ONE/a[lda*k+k];
416             dscal_ur(n-(k+1),t,&a[lda*k+k+1],1);
417
418             /* row elimination with column indexing */
419
420             for (j = kp1; j < n; j++)
421             {
422                 t = a[lda*j+1];
423                 if (l != k)
424                 {
425                     a[lda*j+1] = a[lda*j+k];
426                     a[lda*j+k] = t;
427                 }
428                 daxpy_ur(n-(k+1),t,&a[lda*k+k+1],1,&a[lda*j+k+1],1);
429             }
430         }
431         else
432             (*info) = k;
433     }
434     ipvt[n-1] = n-1;
435     if (a[lda*(n-1)+(n-1)] == ZERO)
436         (*info) = n-1;
437 }
438 }
439
440
441 /*
442 **
443 ** DGESL benchmark
444 **
445 ** We would like to declare a[][lda], but c does not allow it. In this
446 ** function, references to a[i][j] are written a[lda*i+j].
447 **
448 ** dgesl solves the double precision system
449 ** a * x = b or trans(a) * x = b
450 ** using the factors computed by dgeco or dgefa.
451 **
452 ** on entry
453 **
454 **     a       double precision[n][lda]
455 **             the output from dgeco or dgefa.
456 **
457 **     lda     integer
458 **             the leading dimension of the array a .
459 **
460 **     n       integer

```

```

461 **           the order of the matrix  a .
462 **
463 **   ipvt     integer[n]
464 **           the pivot vector from dgeco or dgefa.
465 **
466 **   b       double precision[n]
467 **           the right hand side vector.
468 **
469 **   job     integer
470 **           = 0           to solve  a*x = b ,
471 **           = nonzero    to solve  trans(a)*x = b  where
472 **                       trans(a)  is the transpose.
473 **
474 ** on return
475 **
476 **   b       the solution vector  x .
477 **
478 ** error condition
479 **
480 **   a division by zero will occur if the input factor contains a
481 **   zero on the diagonal.  technically this indicates singularity
482 **   but it is often caused by improper arguments or improper
483 **   setting of lda .  it will not occur if the subroutines are
484 **   called correctly and if dgeco has set rcond .gt. 0.0
485 **   or dgefa has set info .eq. 0 .
486 **
487 ** to compute inverse(a) * c  where  c  is a matrix
488 ** with  p  columns
489 **   dgeco(a,lda,n,ipvt,rcond,z)
490 **   if (!rcond is too small){
491 **       for (j=0,j<p,j++){
492 **           dgesl(a,lda,n,ipvt,c[j][0],0);
493 **       }
494 **
495 ** linpack. this version dated 08/14/78 .
496 ** cleve moler, university of new mexico, argonne national lab.
497 **
498 ** functions
499 **
500 ** blas daxpy,ddot
501 */
502 static void dgesl(REAL *a,int lda,int n,int *ipvt,REAL *b,int job,int roll)
503
504 {
505     REAL    t;
506     int     k,kb,l,nm1;
507
508     if (roll)
509     {
510         nm1 = n - 1;
511         if (job == 0)
512         {
513
514             /* job = 0 , solve  a * x = b  */
515             /* first solve  l*y = b      */
516
517             if (nm1 >= 1)
518                 for (k = 0; k < nm1; k++)
519                 {
520                     l = ipvt[k];
521                     t = b[l];
522                     if (l != k)
523                     {
524                         b[l] = b[k];
525                         b[k] = t;
526                     }

```

```

527         daxpy_r(n-(k+1),t,&a[lda*k+k+1],1,&b[k+1],1);
528     }
529
530     /* now solve u*x = y */
531
532     for (kb = 0; kb < n; kb++)
533     {
534         k = n - (kb + 1);
535         b[k] = b[k]/a[lda*k+k];
536         t = -b[k];
537         daxpy_r(k,t,&a[lda*k+0],1,&b[0],1);
538     }
539 }
540 else
541 {
542
543     /* job = nonzero, solve trans(a) * x = b */
544     /* first solve trans(u)*y = b */
545
546     for (k = 0; k < n; k++)
547     {
548         t = ddot_r(k,&a[lda*k+0],1,&b[0],1);
549         b[k] = (b[k] - t)/a[lda*k+k];
550     }
551
552     /* now solve trans(l)*x = y */
553
554     if (nm1 >= 1)
555         for (kb = 1; kb < nm1; kb++)
556         {
557             k = n - (kb+1);
558             b[k] = b[k] + ddot_r(n-(k+1),&a[lda*k+k+1],1,&b[k+1],1);
559             l = ipvt[k];
560             if (l != k)
561             {
562                 t = b[l];
563                 b[l] = b[k];
564                 b[k] = t;
565             }
566         }
567     }
568 }
569 else
570 {
571     nm1 = n - 1;
572     if (job == 0)
573     {
574
575         /* job = 0 , solve a * x = b */
576         /* first solve l*y = b */
577
578         if (nm1 >= 1)
579             for (k = 0; k < nm1; k++)
580             {
581                 l = ipvt[k];
582                 t = b[l];
583                 if (l != k)
584                 {
585                     b[l] = b[k];
586                     b[k] = t;
587                 }
588                 daxpy_ur(n-(k+1),t,&a[lda*k+k+1],1,&b[k+1],1);
589             }
590
591         /* now solve u*x = y */
592

```

```

593         for (kb = 0; kb < n; kb++)
594             {
595                 k = n - (kb + 1);
596                 b[k] = b[k]/a[lda*k+k];
597                 t = -b[k];
598                 daxpy_ur(k,t,&a[lda*k+0],1,&b[0],1);
599             }
600     }
601     else
602     {
603
604         /* job = nonzero, solve trans(a) * x = b */
605         /* first solve trans(u)*y = b */
606
607         for (k = 0; k < n; k++)
608             {
609                 t = ddot_ur(k,&a[lda*k+0],1,&b[0],1);
610                 b[k] = (b[k] - t)/a[lda*k+k];
611             }
612
613         /* now solve trans(l)*x = y */
614
615         if (nm1 >= 1)
616             for (kb = 1; kb < nm1; kb++)
617                 {
618                     k = n - (kb+1);
619                     b[k] = b[k] + ddot_ur(n-(k+1),&a[lda*k+k+1],1,&b[k+1],1);
620                     l = ipvt[k];
621                     if (l != k)
622                         {
623                             t = b[l];
624                             b[l] = b[k];
625                             b[k] = t;
626                         }
627                 }
628     }
629 }
630
631
632
633
634 /*
635 ** Constant times a vector plus a vector.
636 ** Jack Dongarra, linpack, 3/11/78.
637 ** ROLLED version
638 */
639 static void daxpy_r(int n,REAL da,REAL *dx,int incx,REAL *dy,int incy)
640 {
641     int i,ix,iy;
642
643     if (n <= 0)
644         return;
645     if (da == ZERO)
646         return;
647
648     if (incx != 1 || incy != 1)
649     {
650         {
651
652             /* code for unequal increments or equal increments != 1 */
653
654             ix = 1;
655             iy = 1;
656             if(incx < 0) ix = (-n+1)*incx + 1;
657             if(incy < 0) iy = (-n+1)*incy + 1;
658             for (i = 0;i < n; i++)

```

```

659         {
660         dy[iy] = dy[iy] + da*dx[ix];
661         ix = ix + incx;
662         iy = iy + incy;
663         }
664     return;
665 }
666
667 /* code for both increments equal to 1 */
668
669 for (i = 0; i < n; i++)
670     dy[i] = dy[i] + da*dx[i];
671 }
672
673
674 /*
675 ** Forms the dot product of two vectors.
676 ** Jack Dongarra, linpack, 3/11/78.
677 ** ROLLED version
678 */
679 static REAL ddot_r(int n, REAL *dx, int incx, REAL *dy, int incy)
680
681 {
682     REAL dtemp;
683     int i, ix, iy;
684
685     dtemp = ZERO;
686
687     if (n <= 0)
688         return(ZERO);
689
690     if (incx != 1 || incy != 1)
691     {
692
693         /* code for unequal increments or equal increments != 1 */
694
695         ix = 0;
696         iy = 0;
697         if (incx < 0) ix = (-n+1)*incx;
698         if (incy < 0) iy = (-n+1)*incy;
699         for (i = 0; i < n; i++)
700         {
701             dtemp = dtemp + dx[ix]*dy[iy];
702             ix = ix + incx;
703             iy = iy + incy;
704         }
705         return(dtemp);
706     }
707
708     /* code for both increments equal to 1 */
709
710     for (i=0; i < n; i++)
711         dtemp = dtemp + dx[i]*dy[i];
712     return(dtemp);
713 }
714
715
716 /*
717 ** Scales a vector by a constant.
718 ** Jack Dongarra, linpack, 3/11/78.
719 ** ROLLED version
720 */
721 static void dscal_r(int n, REAL da, REAL *dx, int incx)
722
723 {
724     int i, nincx;

```

```

725
726     if (n <= 0)
727         return;
728     if (incx != 1)
729     {
730
731         /* code for increment not equal to 1 */
732
733         nincx = n*incx;
734         for (i = 0; i < nincx; i = i + incx)
735             dx[i] = da*dx[i];
736         return;
737     }
738
739     /* code for increment equal to 1 */
740
741     for (i = 0; i < n; i++)
742         dx[i] = da*dx[i];
743 }
744
745
746 /*
747 ** constant times a vector plus a vector.
748 ** Jack Dongarra, linpack, 3/11/78.
749 ** UNROLLED version
750 */
751 static void daxpy_ur(int n,REAL da,REAL *dx,int incx,REAL *dy,int incy)
752
753 {
754     int i,ix,iy,m;
755
756     if (n <= 0)
757         return;
758     if (da == ZERO)
759         return;
760
761     if (incx != 1 || incy != 1)
762     {
763
764         /* code for unequal increments or equal increments != 1 */
765
766         ix = 1;
767         iy = 1;
768         if(incx < 0) ix = (-n+1)*incx + 1;
769         if(incy < 0) iy = (-n+1)*incy + 1;
770         for (i = 0; i < n; i++)
771         {
772             dy[iy] = dy[iy] + da*dx[ix];
773             ix = ix + incx;
774             iy = iy + incy;
775         }
776         return;
777     }
778
779     /* code for both increments equal to 1 */
780
781     m = n % 4;
782     if ( m != 0)
783     {
784         for (i = 0; i < m; i++)
785             dy[i] = dy[i] + da*dx[i];
786         if (n < 4)
787             return;
788     }
789     for (i = m; i < n; i = i + 4)
790     {

```



```

791     dy[i] = dy[i] + da*dx[i];
792     dy[i+1] = dy[i+1] + da*dx[i+1];
793     dy[i+2] = dy[i+2] + da*dx[i+2];
794     dy[i+3] = dy[i+3] + da*dx[i+3];
795 }
796 }
797
798
799 /*
800 ** Forms the dot product of two vectors.
801 ** Jack Dongarra, linpack, 3/11/78.
802 ** UNROLLED version
803 */
804 static REAL ddot_ur(int n,REAL *dx,int incx,REAL *dy,int incy)
805
806 {
807     REAL dtemp;
808     int i,ix,iy,m;
809
810     dtemp = ZERO;
811
812     if (n <= 0)
813         return(ZERO);
814
815     if (incx != 1 || incy != 1)
816     {
817
818         /* code for unequal increments or equal increments != 1 */
819
820         ix = 0;
821         iy = 0;
822         if (incx < 0) ix = (-n+1)*incx;
823         if (incy < 0) iy = (-n+1)*incy;
824         for (i = 0; i < n; i++)
825         {
826             dtemp = dtemp + dx[ix]*dy[iy];
827             ix = ix + incx;
828             iy = iy + incy;
829         }
830         return(dtemp);
831     }
832
833     /* code for both increments equal to 1 */
834
835     m = n % 5;
836     if (m != 0)
837     {
838         for (i = 0; i < m; i++)
839             dtemp = dtemp + dx[i]*dy[i];
840         if (n < 5)
841             return(dtemp);
842     }
843     for (i = m; i < n; i = i + 5)
844     {
845         dtemp = dtemp + dx[i]*dy[i] +
846             dx[i+1]*dy[i+1] + dx[i+2]*dy[i+2] +
847             dx[i+3]*dy[i+3] + dx[i+4]*dy[i+4];
848     }
849     return(dtemp);
850 }
851
852
853 /*
854 ** Scales a vector by a constant.
855 ** Jack Dongarra, linpack, 3/11/78.
856 ** UNROLLED version

```

```

857 */
858 static void dscal_ur(int n,REAL da,REAL *dx,int incx)
859
860 {
861     int i,m,nincx;
862
863     if (n <= 0)
864         return;
865     if (incx != 1)
866     {
867
868         /* code for increment not equal to 1 */
869
870         nincx = n*incx;
871         for (i = 0; i < nincx; i = i + incx)
872             dx[i] = da*dx[i];
873         return;
874     }
875
876     /* code for increment equal to 1 */
877
878     m = n % 5;
879     if (m != 0)
880     {
881         for (i = 0; i < m; i++)
882             dx[i] = da*dx[i];
883         if (n < 5)
884             return;
885     }
886     for (i = m; i < n; i = i + 5)
887     {
888         dx[i] = da*dx[i];
889         dx[i+1] = da*dx[i+1];
890         dx[i+2] = da*dx[i+2];
891         dx[i+3] = da*dx[i+3];
892         dx[i+4] = da*dx[i+4];
893     }
894 }
895
896
897 /*
898 ** Finds the index of element having max. absolute value.
899 ** Jack Dongarra, linpack, 3/11/78.
900 */
901 static int idamax(int n,REAL *dx,int incx)
902
903 {
904     REAL dmax;
905     int i, ix, itemp;
906
907     if (n < 1)
908         return(-1);
909     if (n ==1 )
910         return(0);
911     if(incx != 1)
912     {
913
914         /* code for increment not equal to 1 */
915
916         ix = 1;
917         dmax = fabs((double)dx[0]);
918         ix = ix + incx;
919         for (i = 1; i < n; i++)
920         {
921             if(fabs((double)dx[ix]) > dmax)
922                 {

```

```

923         itemp = i;
924         dmax = fabs((double)dx[ix]);
925     }
926     ix = ix + incx;
927 }
928 }
929 else
930 {
931
932     /* code for increment equal to 1 */
933
934     itemp = 0;
935     dmax = fabs((double)dx[0]);
936     for (i = 1; i < n; i++)
937         if(fabs((double)dx[i]) > dmax)
938             {
939                 itemp = i;
940                 dmax = fabs((double)dx[i]);
941             }
942 }
943 return (itemp);
944 }
945
946 static REAL second(void)
947 {
948     REAL help;
949     bzero(buffer,256);
950     sprintf(buffer,"time\n");
951     n = write(sockfd,buffer,strlen(buffer));
952     if (n < 0)
953         error("ERROR writing to socket");
954     bzero(buffer,256);
955     n = read(sockfd,buffer,255);
956     if (n < 0)
957         error("ERROR reading from socket");
958     help = atof(buffer);
959     help /= 1000000;
960
961     bzero(buffer,256);
962     return help;
963 }
964

```

## Linpack-Client-Windows

Listing 5.3: Linpack-Client-Windows

```

1  /*
2  **
3  ** To compile: cc -O -o linpackclient linpackclient.c -lm
4  **
5  */
6  #pragma comment( lib, "ws2_32.lib" )
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <math.h>
11 #include <time.h>
12 #include <float.h>
13
14 #include <sys/types.h>
15 #include <windows.h>
16 #include <winsock.h>
17

```

```

18
19 #define DP
20
21 #ifndef SP
22 #define ZERO          0.0
23 #define ONE          1.0
24 #define PREC         "Single"
25 #define BASE10DIG    FLT_DIG
26
27 typedef float   REAL;
28 #endif
29
30 #ifdef DP
31 #define ZERO          0.0e0
32 #define ONE          1.0e0
33 #define PREC         "Double"
34 #define BASE10DIG    DBL_DIG
35
36 typedef double  REAL;
37 #endif
38
39 static REAL linpack (long nreps, int arsize);
40 static void matgen  (REAL *a, int lda, int n, REAL *b, REAL *norma);
41 static void dgefa   (REAL *a, int lda, int n, int *ipvt, int *info, int roll);
42 static void dgesl   (REAL *a, int lda, int n, int *ipvt, REAL *b, int job, int roll);
43 static void daxpy_r (int n, REAL da, REAL *dx, int incx, REAL *dy, int incy);
44 static REAL ddot_r  (int n, REAL *dx, int incx, REAL *dy, int incy);
45 static void dscal_r (int n, REAL da, REAL *dx, int incx);
46 static void daxpy_ur (int n, REAL da, REAL *dx, int incx, REAL *dy, int incy);
47 static REAL ddot_ur (int n, REAL *dx, int incx, REAL *dy, int incy);
48 static void dscal_ur (int n, REAL da, REAL *dx, int incx);
49 static int idamax  (int n, REAL *dx, int incx);
50 static REAL second (void);
51 int startWinsock(void);
52
53
54 static void *mempool;
55
56 // socket
57 int sockfd, portno, n;
58 struct sockaddr_in serv_addr;
59 struct hostent *server;
60 char buffer[256];
61 SOCKET s;
62
63
64
65 int main(int argc, char *argv[])
66 {
67     {
68         char    buf[80];
69         int     arsize, i;
70         long    arsize2d, memreq, nreps;
71         size_t  malloc_arg;
72
73
74         // Create socket and connect to server
75         long rc;
76         SOCKADDR_IN addr;
77
78         rc = startWinsock();
79         if (rc != 0)
80         {
81             printf("Fehler: startWinsock, fehler code: %d\n", rc);
82             return 1;
83         }
84     }

```

```

84     else
85     {
86         printf("Winsock gestartet!\n");
87     }
88
89     s=socket(AF_INET,SOCK_STREAM,0);
90     if (s==INVALID_SOCKET)
91     {
92         printf("Fehler: Der Socket konnte nicht erstellt werden, fehler code: %d\n",
93             WSAGetLastError());
94         return 1;
95     }
96     else
97     {
98         printf("Socket erstellt!\n");
99     }
100
101
102     if (argc < 3)
103     {
104         fprintf(stderr,"usage %s hostname port\n", argv[0]);
105         exit(0);
106     }
107     portno = atoi(argv[2]);
108
109     server = gethostbyname(argv[1]);
110
111
112     if (server == NULL)
113     {
114         fprintf(stderr,"ERROR, no such host\n");
115         exit(0);
116     }
117
118     memset(&addr,0,sizeof(SOCKADDR_IN));
119     addr.sin_family=AF_INET;
120     addr.sin_port=htons(portno);
121     addr.sin_addr.s_addr=inet_addr(argv[1]);
122     rc=connect(s,(SOCKADDR*)&addr,sizeof(SOCKADDR));
123     if (rc==SOCKET_ERROR)
124     {
125         printf("Fehler: connect gescheitert, fehler code: %d\n",WSAGetLastError());
126         return 1;
127     }
128     else
129     {
130         printf("Verbunden mit %s\n",argv[1]);
131     }
132
133
134
135
136     for (i=1000; i<8001; i*=2)
137     {
138         //printf("Enter array size (q to quit) [200]: ");
139         //fgets(buf,79,stdin);
140         //if (buf[0]=='q' || buf[0]=='Q')
141         //    break;
142         //if (buf[0]=='\0' || buf[0]=='\n')
143         //    arsize=200;
144         //else
145         //    arsize=atoi(buf);
146         arsize = i;
147         arsize/=2;
148         arsize*=2;

```

```

149     if (arsize<10)
150     {
151         printf("Too small.\n");
152         continue;
153     }
154     arsize2d = (long)arsize*(long)arsize;
155     memreq=arsize2d*sizeof(REAL)+(long)arsize*sizeof(REAL)+(long)arsize*sizeof(
156         int);
157     printf("Memory required: %ldK.\n",(memreq+512L)>>10);
158     malloc_arg=(size_t)memreq;
159     if (malloc_arg!=memreq || (mempool=malloc(malloc_arg))==NULL)
160     {
161         printf("Not enough memory available for given array size.\n\n");
162         continue;
163     }
164     printf("\n\nLINPACK benchmark, %s precision.\n",PREC);
165     printf("Machine precision: %d digits.\n",BASE10DIG);
166     printf("Array size %d X %d.\n",arsize,arsize);
167     printf("Average rolled and unrolled performance:\n\n");
168     printf("      Reps Time(s) DGEFA  DGESL  OVERHEAD   KFLOPS\n");
169     printf("-----\n");
170     nreps=1;
171
172     memset(buffer,0,256);
173     sprintf(buffer,"start\n");
174     n = send(s,buffer,strlen(buffer),0);
175     if (n < 0)
176         printf("ERROR writing to socket\n");
177     memset(buffer,0,256);
178     n = recv(s,buffer,255,0);
179     if (n < 0)
180         printf("ERROR reading from socket\n");
181     if (strcmp(buffer,"start") != 0)
182     {
183         printf("ERROR from server\n");
184     }
185
186     linpack(10,arsize);
187     free(mempool);
188     printf("\n");
189     }
190
191     memset(buffer,0,256);
192     sprintf(buffer,"quit\n");
193     n = send(s,buffer,strlen(buffer),0);
194     if (n < 0)
195         printf("ERROR writing to socket\n");
196     memset(buffer,0,256);
197     n = recv(s,buffer,255,0);
198     if (n < 0)
199         printf("ERROR reading from socket");
200     if (strcmp(buffer,"quit") == 0)
201     {
202         printf("Quitting linpack\n");
203     }
204     closesocket(s);
205     WSACleanup();
206     return 0;
207 }
208
209 static REAL linpack(long nreps,int arsize)
210 {
211     REAL  *a,*b;
212     REAL  norma,t1,kflops,tdgesl,tdgefa,totalt,toverhead,ops;
213

```

```

214     int    *ipvt,n,info,lda;
215     long   i,arsize2d;
216
217     lda = arsize;
218     n = arsize/2;
219     arsize2d = (long)arsize*(long)arsize;
220     ops=((2.0*n*n*n)/3.0+2.0*n*n);
221     a=(REAL *)mempool;
222     b=a+arsize2d;
223     ipvt=(int *)&b[arsize];
224     tdgesl=0;
225     tdgefa=0;
226
227     totalt=second();
228     for (i=0;i<nreps;i++)
229     {
230         matgen(a,lda,n,b,&norma);
231         t1 = second();
232         dgefa(a,lda,n,ipvt,&info,1);
233         tdgefa += second()-t1;
234         t1 = second();
235         dgesl(a,lda,n,ipvt,b,0,1);
236         tdgesl += second()-t1;
237     }
238     for (i=0;i<nreps;i++)
239     {
240         matgen(a,lda,n,b,&norma);
241         t1 = second();
242         dgefa(a,lda,n,ipvt,&info,0);
243         tdgefa += second()-t1;
244         t1 = second();
245         dgesl(a,lda,n,ipvt,b,0,0);
246         tdgesl += second()-t1;
247     }
248     totalt=second()-totalt;
249     if (totalt<0.5 || tdgefa+tdgesl<0.2)
250         return(0.);
251     kflops=2.*nreps*ops/(1000.*(tdgefa+tdgesl));
252     toverhead=totalt-tdgefa-tdgesl;
253     if (tdgefa<0.)
254         tdgefa=0.;
255     if (tdgesl<0.)
256         tdgesl=0.;
257     if (toverhead<0.)
258         toverhead=0.;
259     printf("%8ld %6.2f %6.2f%% %6.2f%% %6.2f%% %9.3f\n",
260           nreps,totalt,100.*tdgefa/totalt,
261           100.*tdgesl/totalt,100.*toverhead/totalt,
262           kflops);
263     return(totalt);
264 }
265
266
267 /*
268 ** For matgen,
269 ** We would like to declare a[][lda], but c does not allow it. In this
270 ** function, references to a[i][j] are written a[lda*i+j].
271 */
272 static void matgen(REAL *a,int lda,int n,REAL *b,REAL *norma)
273
274 {
275     int init,i,j;
276
277     init = 1325;
278     *norma = 0.0;
279     for (j = 0; j < n; j++)

```

```

280     for (i = 0; i < n; i++)
281     {
282         init = (int)((long)3125*(long)init % 65536L);
283         a[lda*j+i] = (init - 32768.0)/16384.0;
284         *norma = (a[lda*j+i] > *norma) ? a[lda*j+i] : *norma;
285     }
286     for (i = 0; i < n; i++)
287         b[i] = 0.0;
288     for (j = 0; j < n; j++)
289         for (i = 0; i < n; i++)
290             b[i] = b[i] + a[lda*j+i];
291 }
292
293
294 /*
295 **
296 ** DGEFA benchmark
297 **
298 ** We would like to declare a[][lda], but c does not allow it. In this
299 ** function, references to a[i][j] are written a[lda*i+j].
300 **
301 ** dgefa factors a double precision matrix by gaussian elimination.
302 **
303 ** dgefa is usually called by dgeco, but it can be called
304 ** directly with a saving in time if rcond is not needed.
305 ** (time for dgeco) = (1 + 9/n)*(time for dgefa) .
306 **
307 ** on entry
308 **
309 **     a       REAL precision[n][lda]
310 **           the matrix to be factored.
311 **
312 **     lda     integer
313 **           the leading dimension of the array a .
314 **
315 **     n       integer
316 **           the order of the matrix a .
317 **
318 ** on return
319 **
320 **     a       an upper triangular matrix and the multipliers
321 **           which were used to obtain it.
322 **           the factorization can be written a = l*u where
323 **           l is a product of permutation and unit lower
324 **           triangular matrices and u is upper triangular.
325 **
326 **     ipvt    integer[n]
327 **           an integer vector of pivot indices.
328 **
329 **     info    integer
330 **           = 0 normal value.
331 **           = k if u[k][k].eq. 0.0 . this is not an error
332 **           condition for this subroutine, but it does
333 **           indicate that dgesl or dgedi will divide by zero
334 **           if called. use rcond in dgeco for a reliable
335 **           indication of singularity.
336 **
337 ** linpack. this version dated 08/14/78 .
338 ** cleve moler, university of New Mexico, argonne national lab.
339 **
340 ** functions
341 **
342 ** blas daxpy,dscal,idamax
343 **
344 */
345 static void dgefa(REAL *a,int lda,int n,int *ipvt,int *info,int roll)

```



```

346
347 {
348 REAL t;
349 int idamax(),j,k,kp1,l,nm1;
350
351 /* gaussian elimination with partial pivoting */
352
353 if (roll)
354 {
355     *info = 0;
356     nm1 = n - 1;
357     if (nm1 >= 0)
358         for (k = 0; k < nm1; k++)
359             {
360                 kp1 = k + 1;
361
362                 /* find l = pivot index */
363
364                 l = idamax(n-k,&a[lda*k+k],1) + k;
365                 ipvt[k] = l;
366
367                 /* zero pivot implies this column already
368                  triangularized */
369
370                 if (a[lda*k+1] != ZERO)
371                     {
372
373                         /* interchange if necessary */
374
375                         if (l != k)
376                             {
377                                 t = a[lda*k+1];
378                                 a[lda*k+1] = a[lda*k+k];
379                                 a[lda*k+k] = t;
380                             }
381
382                         /* compute multipliers */
383
384                         t = -ONE/a[lda*k+k];
385                         dscal_r(n-(k+1),t,&a[lda*k+k+1],1);
386
387                         /* row elimination with column indexing */
388
389                         for (j = kp1; j < n; j++)
390                             {
391                                 t = a[lda*j+1];
392                                 if (l != k)
393                                     {
394                                         a[lda*j+1] = a[lda*j+k];
395                                         a[lda*j+k] = t;
396                                     }
397                                 daxpy_r(n-(k+1),t,&a[lda*k+k+1],1,&a[lda*j+k+1],1);
398                             }
399                     }
400                 else
401                     (*info) = k;
402             }
403     ipvt[n-1] = n-1;
404     if (a[lda*(n-1)+(n-1)] == ZERO)
405         (*info) = n-1;
406 }
407 else
408 {
409     *info = 0;
410     nm1 = n - 1;
411     if (nm1 >= 0)

```

```

412     for (k = 0; k < nm1; k++)
413     {
414         kp1 = k + 1;
415
416         /* find l = pivot index */
417
418         l = idamax(n-k,&a[lda*k+k],1) + k;
419         ipvt[k] = l;
420
421         /* zero pivot implies this column already
422            triangularized */
423
424         if (a[lda*k+1] != ZERO)
425         {
426
427             /* interchange if necessary */
428
429             if (l != k)
430             {
431                 t = a[lda*k+1];
432                 a[lda*k+1] = a[lda*k+k];
433                 a[lda*k+k] = t;
434             }
435
436             /* compute multipliers */
437
438             t = -ONE/a[lda*k+k];
439             dscal_ur(n-(k+1),t,&a[lda*k+k+1],1);
440
441             /* row elimination with column indexing */
442
443             for (j = kp1; j < n; j++)
444             {
445                 t = a[lda*j+1];
446                 if (l != k)
447                 {
448                     a[lda*j+1] = a[lda*j+k];
449                     a[lda*j+k] = t;
450                 }
451                 daxpy_ur(n-(k+1),t,&a[lda*k+k+1],1,&a[lda*j+k+1],1);
452             }
453         }
454         else
455             (*info) = k;
456     }
457     ipvt[n-1] = n-1;
458     if (a[lda*(n-1)+(n-1)] == ZERO)
459         (*info) = n-1;
460 }
461
462
463
464 /*
465 **
466 ** DGESL benchmark
467 **
468 ** We would like to declare a[][lda], but c does not allow it. In this
469 ** function, references to a[i][j] are written a[lda*i+j].
470 **
471 ** dgesl solves the double precision system
472 ** a * x = b or trans(a) * x = b
473 ** using the factors computed by dgeco or dgefa.
474 **
475 ** on entry
476 **
477 **     a         double precision[n][lda]

```

```

478 **           the output from dgeco or dgefa.
479 **
480 **   lda      integer
481 **           the leading dimension of the array  a .
482 **
483 **   n        integer
484 **           the order of the matrix  a .
485 **
486 **   ipvt     integer[n]
487 **           the pivot vector from dgeco or dgefa.
488 **
489 **   b        double precision[n]
490 **           the right hand side vector.
491 **
492 **   job      integer
493 **           = 0          to solve  a*x = b ,
494 **           = nonzero    to solve  trans(a)*x = b  where
495 **                       trans(a) is the transpose.
496 **
497 ** on return
498 **
499 **   b        the solution vector  x .
500 **
501 ** error condition
502 **
503 **   a division by zero will occur if the input factor contains a
504 **   zero on the diagonal.  technically this indicates singularity
505 **   but it is often caused by improper arguments or improper
506 **   setting of lda .  it will not occur if the subroutines are
507 **   called correctly and if dgeco has set rcond .gt. 0.0
508 **   or dgefa has set info .eq. 0 .
509 **
510 ** to compute  inverse(a) * c  where  c  is a matrix
511 ** with  p  columns
512 **   dgeco(a,lda,n,ipvt,rcond,z)
513 **   if (!rcond is too small){
514 **       for (j=0,j<p,j++){
515 **           dgesl(a,lda,n,ipvt,c[j][0],0);
516 **       }
517 **
518 ** linpack. this version dated 08/14/78 .
519 ** cleve moler, university of new mexico, argonne national lab.
520 **
521 ** functions
522 **
523 ** blas daxpy,ddot
524 */
525 static void dgesl(REAL *a,int lda,int n,int *ipvt,REAL *b,int job,int roll)
526
527 {
528     REAL    t;
529     int     k,kb,l,nm1;
530
531     if (roll)
532     {
533         nm1 = n - 1;
534         if (job == 0)
535         {
536
537             /* job = 0 , solve  a * x = b  */
538             /* first solve  l*y = b  */
539
540             if (nm1 >= 1)
541                 for (k = 0; k < nm1; k++)
542                 {
543                     l = ipvt[k];

```

```

544         t = b[l];
545         if (l != k)
546             {
547                 b[l] = b[k];
548                 b[k] = t;
549             }
550         daxpy_r(n-(k+1),t,&a[l*lda+k+1],1,&b[k+1],1);
551     }
552
553     /* now solve u*x = y */
554
555     for (kb = 0; kb < n; kb++)
556     {
557         k = n - (kb + 1);
558         b[k] = b[k]/a[l*lda+k];
559         t = -b[k];
560         daxpy_r(k,t,&a[l*lda+k+0],1,&b[0],1);
561     }
562 }
563 else
564 {
565
566     /* job = nonzero, solve trans(a) * x = b */
567     /* first solve trans(u)*y = b */
568
569     for (k = 0; k < n; k++)
570     {
571         t = ddot_r(k,&a[l*lda+k+0],1,&b[0],1);
572         b[k] = (b[k] - t)/a[l*lda+k];
573     }
574
575     /* now solve trans(l)*x = y */
576
577     if (nm1 >= 1)
578         for (kb = 1; kb < nm1; kb++)
579         {
580             k = n - (kb+1);
581             b[k] = b[k] + ddot_r(n-(k+1),&a[l*lda+k+1],1,&b[k+1],1);
582             l = ipvt[k];
583             if (l != k)
584                 {
585                     t = b[l];
586                     b[l] = b[k];
587                     b[k] = t;
588                 }
589         }
590 }
591 }
592 else
593 {
594     nm1 = n - 1;
595     if (job == 0)
596     {
597
598         /* job = 0 , solve a * x = b */
599         /* first solve l*y = b */
600
601         if (nm1 >= 1)
602             for (k = 0; k < nm1; k++)
603             {
604                 l = ipvt[k];
605                 t = b[l];
606                 if (l != k)
607                     {
608                         b[l] = b[k];
609                         b[k] = t;

```

```

610         }
611         daxpy_ur(n-(k+1),t,&a[lda*k+k+1],1,&b[k+1],1);
612     }
613
614     /* now solve u*x = y */
615
616     for (kb = 0; kb < n; kb++)
617     {
618         k = n - (kb + 1);
619         b[k] = b[k]/a[lda*k+k];
620         t = -b[k];
621         daxpy_ur(k,t,&a[lda*k+0],1,&b[0],1);
622     }
623
624     else
625     {
626
627         /* job = nonzero, solve trans(a) * x = b */
628         /* first solve trans(u)*y = b */
629
630         for (k = 0; k < n; k++)
631         {
632             t = ddot_ur(k,&a[lda*k+0],1,&b[0],1);
633             b[k] = (b[k] - t)/a[lda*k+k];
634         }
635
636         /* now solve trans(l)*x = y */
637
638         if (nm1 >= 1)
639             for (kb = 1; kb < nm1; kb++)
640             {
641                 k = n - (kb+1);
642                 b[k] = b[k] + ddot_ur(n-(k+1),&a[lda*k+k+1],1,&b[k+1],1);
643                 l = ipvt[k];
644                 if (l != k)
645                 {
646                     t = b[l];
647                     b[l] = b[k];
648                     b[k] = t;
649                 }
650             }
651     }
652 }
653 }
654
655
656
657 /*
658 ** Constant times a vector plus a vector.
659 ** Jack Dongarra, linpack, 3/11/78.
660 ** ROLLED version
661 */
662 static void daxpy_r(int n,REAL da,REAL *dx,int incx,REAL *dy,int incy)
663
664 {
665     int i,ix,iy;
666
667     if (n <= 0)
668         return;
669     if (da == ZERO)
670         return;
671
672     if (incx != 1 || incy != 1)
673     {
674
675         /* code for unequal increments or equal increments != 1 */

```

```

676
677     ix = 1;
678     iy = 1;
679     if(incx < 0) ix = (-n+1)*incx + 1;
680     if(incy < 0) iy = (-n+1)*incy + 1;
681     for (i = 0; i < n; i++)
682     {
683         dy[iy] = dy[iy] + da*dx[ix];
684         ix = ix + incx;
685         iy = iy + incy;
686     }
687     return;
688 }
689
690 /* code for both increments equal to 1 */
691
692 for (i = 0; i < n; i++)
693     dy[i] = dy[i] + da*dx[i];
694 }
695
696
697 /*
698 ** Forms the dot product of two vectors.
699 ** Jack Dongarra, linpack, 3/11/78.
700 ** ROLLED version
701 **/
702 static REAL ddot_r(int n, REAL *dx, int incx, REAL *dy, int incy)
703
704 {
705     REAL dtemp;
706     int i, ix, iy;
707
708     dtemp = ZERO;
709
710     if (n <= 0)
711         return(ZERO);
712
713     if (incx != 1 || incy != 1)
714     {
715
716         /* code for unequal increments or equal increments != 1 */
717
718         ix = 0;
719         iy = 0;
720         if (incx < 0) ix = (-n+1)*incx;
721         if (incy < 0) iy = (-n+1)*incy;
722         for (i = 0; i < n; i++)
723         {
724             dtemp = dtemp + dx[ix]*dy[iy];
725             ix = ix + incx;
726             iy = iy + incy;
727         }
728         return(dtemp);
729     }
730
731     /* code for both increments equal to 1 */
732
733     for (i=0; i < n; i++)
734         dtemp = dtemp + dx[i]*dy[i];
735     return(dtemp);
736 }
737
738
739 /*
740 ** Scales a vector by a constant.
741 ** Jack Dongarra, linpack, 3/11/78.

```

```

742 ** ROLLED version
743 */
744 static void dscal_r(int n,REAL da,REAL *dx,int incx)
745 {
746     int i,nincx;
747
748     if (n <= 0)
749         return;
750     if (incx != 1)
751     {
752
753         /* code for increment not equal to 1 */
754
755         nincx = n*incx;
756         for (i = 0; i < nincx; i = i + incx)
757             dx[i] = da*dx[i];
758         return;
759     }
760
761     /* code for increment equal to 1 */
762
763     for (i = 0; i < n; i++)
764         dx[i] = da*dx[i];
765 }
766
767
768
769 /*
770 ** constant times a vector plus a vector.
771 ** Jack Dongarra, linpack, 3/11/78.
772 ** UNROLLED version
773 */
774 static void daxpy_ur(int n,REAL da,REAL *dx,int incx,REAL *dy,int incy)
775 {
776     int i,ix,iy,m;
777
778     if (n <= 0)
779         return;
780     if (da == ZERO)
781         return;
782
783     if (incx != 1 || incy != 1)
784     {
785
786         /* code for unequal increments or equal increments != 1 */
787
788         ix = 1;
789         iy = 1;
790         if(incx < 0) ix = (-n+1)*incx + 1;
791         if(incy < 0) iy = (-n+1)*incy + 1;
792         for (i = 0; i < n; i++)
793         {
794             dy[iy] = dy[iy] + da*dx[ix];
795             ix = ix + incx;
796             iy = iy + incy;
797         }
798         return;
799     }
800
801     /* code for both increments equal to 1 */
802
803     m = n % 4;
804     if (m != 0)
805     {
806         for (i = 0; i < m; i++)

```

```

808         dy[i] = dy[i] + da*dx[i];
809     if (n < 4)
810         return;
811     }
812     for (i = m; i < n; i = i + 4)
813     {
814         dy[i] = dy[i] + da*dx[i];
815         dy[i+1] = dy[i+1] + da*dx[i+1];
816         dy[i+2] = dy[i+2] + da*dx[i+2];
817         dy[i+3] = dy[i+3] + da*dx[i+3];
818     }
819 }
820
821
822 /*
823 ** Forms the dot product of two vectors.
824 ** Jack Dongarra, linpack, 3/11/78.
825 ** UNROLLED version
826 **/
827 static REAL ddot_ur(int n,REAL *dx,int incx,REAL *dy,int incy)
828
829 {
830     REAL dtemp;
831     int i,ix,iy,m;
832
833     dtemp = ZERO;
834
835     if (n <= 0)
836         return(ZERO);
837
838     if (incx != 1 || incy != 1)
839     {
840
841         /* code for unequal increments or equal increments != 1 */
842
843         ix = 0;
844         iy = 0;
845         if (incx < 0) ix = (-n+1)*incx;
846         if (incy < 0) iy = (-n+1)*incy;
847         for (i = 0; i < n; i++)
848         {
849             dtemp = dtemp + dx[ix]*dy[iy];
850             ix = ix + incx;
851             iy = iy + incy;
852         }
853         return(dtemp);
854     }
855
856     /* code for both increments equal to 1 */
857
858     m = n % 5;
859     if (m != 0)
860     {
861         for (i = 0; i < m; i++)
862             dtemp = dtemp + dx[i]*dy[i];
863         if (n < 5)
864             return(dtemp);
865     }
866     for (i = m; i < n; i = i + 5)
867     {
868         dtemp = dtemp + dx[i]*dy[i] +
869             dx[i+1]*dy[i+1] + dx[i+2]*dy[i+2] +
870             dx[i+3]*dy[i+3] + dx[i+4]*dy[i+4];
871     }
872     return(dtemp);
873 }

```



```

874
875
876 /*
877 ** Scales a vector by a constant.
878 ** Jack Dongarra, linpack, 3/11/78.
879 ** UNROLLED version
880 */
881 static void dscal_ur(int n,REAL da,REAL *dx,int incx)
882
883 {
884     int i,m,nincx;
885
886     if (n <= 0)
887         return;
888     if (incx != 1)
889     {
890
891         /* code for increment not equal to 1 */
892
893         nincx = n*incx;
894         for (i = 0; i < nincx; i = i + incx)
895             dx[i] = da*dx[i];
896         return;
897     }
898
899     /* code for increment equal to 1 */
900
901     m = n % 5;
902     if (m != 0)
903     {
904         for (i = 0; i < m; i++)
905             dx[i] = da*dx[i];
906         if (n < 5)
907             return;
908     }
909     for (i = m; i < n; i = i + 5)
910     {
911         dx[i] = da*dx[i];
912         dx[i+1] = da*dx[i+1];
913         dx[i+2] = da*dx[i+2];
914         dx[i+3] = da*dx[i+3];
915         dx[i+4] = da*dx[i+4];
916     }
917 }
918
919
920 /*
921 ** Finds the index of element having max. absolute value.
922 ** Jack Dongarra, linpack, 3/11/78.
923 */
924 static int idamax(int n,REAL *dx,int incx)
925
926 {
927     REAL dmax;
928     int i, ix, itemp;
929
930     if (n < 1)
931         return(-1);
932     if (n == 1)
933         return(0);
934     if(incx != 1)
935     {
936
937         /* code for increment not equal to 1 */
938
939         ix = 1;

```

```

940     dmax = fabs((double)dx[0]);
941     ix = ix + incx;
942     for (i = 1; i < n; i++)
943     {
944         if(fabs((double)dx[ix]) > dmax)
945         {
946             itemp = i;
947             dmax = fabs((double)dx[ix]);
948         }
949         ix = ix + incx;
950     }
951 }
952 else
953 {
954
955     /* code for increment equal to 1 */
956
957     itemp = 0;
958     dmax = fabs((double)dx[0]);
959     for (i = 1; i < n; i++)
960         if(fabs((double)dx[i]) > dmax)
961         {
962             itemp = i;
963             dmax = fabs((double)dx[i]);
964         }
965     }
966     return (itemp);
967 }
968
969
970 static REAL second(void)
971 {
972     REAL help;
973     memset(buffer,0,256);
974     sprintf(buffer,"time\n");
975     n = send(s,buffer,strlen(buffer),0);
976     if (n < 0)
977         printf("ERROR writing to socket\n");
978     memset(buffer,0,256);
979     n = recv(s,buffer,255,0);
980     if (n < 0)
981         printf("ERROR reading from socket\n");
982     help = atof(buffer);
983     help /= 1000000;
984
985     memset(buffer,0,256);
986     return help;
987 }
988
989 int startWinsock(void)
990 {
991     WSADATA wsa;
992     return WSAStartup(MAKEWORD(2,0),&wsa);
993 }
994 }

```

## Iometer

Listing 5.4: Iometer-Patch

```

1 Index: src/IOCompletionQ.cpp
2 =====
3 --- src.orig/IOCompletionQ.cpp
4 +++ src/IOCompletionQ.cpp

```

```

5 @@ -319,6 +319,20 @@ BOOL GetQueuedCompletionStatus(HANDLE cq
6 // have to considere changes there as well.
7 SetLastError(cqid->element_list[i].error);
8 return (FALSE);
9 + } else if (cqid->element_list[i].error != 0) {
10 + // Sadly, some systems overload the "read"
11 + // return with the (positive) error value.
12 + // Checking the explicit "error" value is
13 + // a more reliable way to distinguish an
14 + // actual error. Typical errors are
15 + // 104: connection reset by peer
16 + // 32: broken pipe.
17 + // Note that it is important that ReadFile()
18 + // and WriteFile() preset this error value
19 + // to 0 when starting an async IO.
20 + *bytes_transferred = 0;
21 + SetLastError(cqid->element_list[i].error);
22 + return (FALSE);
23 } else {
24 return (TRUE);
25 }
26 @@ -547,6 +561,7 @@ BOOL ReadFile(HANDLE file_handle, void *
27
28 aiocbp = &this_cq->element_list[free_index].aiocbp;
29
30 + memset(aiocbp, 0, sizeof(*aiocbp));
31 aiocbp->aio_buf = buffer;
32 aiocbp->aio_fildes = filep->fd;
33 aiocbp->aio_nbytes = bytes_to_read;
34 @@ -558,6 +573,7 @@ BOOL ReadFile(HANDLE file_handle, void *
35 this_cq->element_list[free_index].data = lpOverlapped;
36 this_cq->element_list[free_index].bytes_transferred = 0;
37 this_cq->element_list[free_index].completion_key =
38 filep->completion_key;
39 + this_cq->element_list[free_index].error = 0;
40
41 *bytes_read = 0;
42
43 @@ -654,6 +670,7 @@ BOOL WriteFile(HANDLE file_handle, void
44
45 aiocbp = &this_cq->element_list[free_index].aiocbp;
46
47 + memset(aiocbp, 0, sizeof(*aiocbp));
48 aiocbp->aio_buf = buffer;
49 aiocbp->aio_fildes = filep->fd;
50 aiocbp->aio_nbytes = bytes_to_write;
51 @@ -665,6 +682,7 @@ BOOL WriteFile(HANDLE file_handle, void
52 this_cq->element_list[free_index].data = lpOverlapped;
53 this_cq->element_list[free_index].bytes_transferred = 0;
54 this_cq->element_list[free_index].completion_key =
55 filep->completion_key;
56 + this_cq->element_list[free_index].error = 0;
57
58 *bytes_written = 0;
59
60 Index: src/IOGrunt.cpp
61 =====
62 --- src.orig/IOGrunt.cpp
63 +++ src/IOGrunt.cpp
64 @@ -1098,6 +1098,7 @@ void Grunt::Do_IOS()
65 Target *target;
66 Raw_Result *target_results; // Pointer to results for selected target.
67 Raw_Result *prev_target_results;
68 + draining_ios = FALSE;
69
70 while (grunt_state != TestIdle) {

```

```

71 #if defined(IOMTR_OSFAMILY_NETWARE)
72 @@ -1336,6 +1337,7 @@ void Grunt::Do_IOs()
73 } // while grunt_state is not TestIdle
74
75 // Drain any outstanding I/Os from the completion queue
76 + draining_ios = TRUE;
77 while (outstanding_ios > 0) {
78 #if defined(IOMTR_OSFAMILY_NETWARE)
79 pthread_yield(); // NetWare is non-preemptive
80 @@ -1366,8 +1368,15 @@ ReturnVal Grunt::Complete_IO(int timeout
81 switch (io_cq->GetStatus(&bytes, &trans_id, timeout)) {
82 case ReturnSuccess:
83 // I/O completed. Make sure we received everything we requested.
84 - if (bytes < (int)trans_slots[trans_id].size)
85 - Do_Partial_IO(&trans_slots[trans_id], bytes);
86 + if (bytes < (int)trans_slots[trans_id].size) {
87 + if (!draining_ios) {
88 + Do_Partial_IO(&trans_slots[trans_id], bytes);
89 + } else {
90 + // We're draining outstanding IOs, so
91 + // don't initiate a new one.
92 + Record_IO(&trans_slots[trans_id], 0);
93 + }
94 + }
95 else
96 Record_IO(&trans_slots[trans_id], timer_value());
97 return ReturnSuccess;
98 Index: src/IOGrunt.h
99 =====
100 --- src.orig/IOGrunt.h
101 +++ src/IOGrunt.h
102 @@ -196,6 +196,7 @@ class Grunt {
103 int available_head;
104 int available_tail;
105 int outstanding_ios;
106 + BOOL draining_ios;
107 //
108 // Operations on related I/O transaction arrays.
109 void Initialize_Transaction_Arrays();
110
111
112 Index: src/IOPerformance.h
113 =====
114 --- src.orig/IOPerformance.h
115 +++ src/IOPerformance.h
116 @@ -97,7 +97,7 @@
117 #include <net/if.h>
118 #endif
119
120 -#if defined(IOMTR_OS_LINUX) || defined(IOMTR_OSFAMILY_NETWARE) ||
121 defined(IOMTR_OS_SOLARIS)
122 +#if defined(IOMTR_OSFAMILY_NETWARE) || defined(IOMTR_OS_SOLARIS)
123 #include <stropts.h>
124 #endif

```

## RAMspeed

Listing 5.5: RAMspeed-Skript

```

1 #!/bin/sh
2 OUTPUT_FILE=$1
3 echo "\nINTmark [writing]"
4 echo "===== " > $OUTPUT_FILE
5 echo "\nINTmark [writing]" >> $OUTPUT_FILE

```

```

6 echo "======" >> $OUTPUT_FILE
7 sync
8 date >> $OUTPUT_FILE
9 ./ramspeed32 -b 1 | tee -a $OUTPUT_FILE
10 date >> $OUTPUT_FILE
11 echo "\nINTmark [reading]"
12 echo "======" >> $OUTPUT_FILE
13 echo "\nINTmark [reading]" >> $OUTPUT_FILE
14 echo "======" >> $OUTPUT_FILE
15 sync
16 date >> $OUTPUT_FILE
17 ./ramspeed32 -b 2 | tee -a $OUTPUT_FILE
18 date >> $OUTPUT_FILE
19 echo "\nINTmem"
20 echo "======" >> $OUTPUT_FILE
21 echo "\nINTmem" >> $OUTPUT_FILE
22 echo "======" >> $OUTPUT_FILE
23 sync
24 date >> $OUTPUT_FILE
25 ./ramspeed32 -b 3 | tee -a $OUTPUT_FILE
26 date >> $OUTPUT_FILE
27 echo "\nFLOATmark [writing]"
28 echo "======" >> $OUTPUT_FILE
29 echo "\nFLOATmark [writing]" >> $OUTPUT_FILE
30 echo "======" >> $OUTPUT_FILE
31 sync
32 date >> $OUTPUT_FILE
33 ./ramspeed32 -b 4 | tee -a $OUTPUT_FILE
34 date >> $OUTPUT_FILE
35 echo "\nFLOATmark [reading]"
36 echo "======" >> $OUTPUT_FILE
37 echo "\nFLOATmark [reading]" >> $OUTPUT_FILE
38 echo "======" >> $OUTPUT_FILE
39 sync
40 date >> $OUTPUT_FILE
41 ./ramspeed32 -b 5 | tee -a $OUTPUT_FILE
42 date >> $OUTPUT_FILE
43 echo "\nFLOATmem"
44 echo "======" >> $OUTPUT_FILE
45 echo "\nFLOATmem" >> $OUTPUT_FILE
46 echo "======" >> $OUTPUT_FILE
47 sync
48 date >> $OUTPUT_FILE
49 ./ramspeed32 -b 6 | tee -a $OUTPUT_FILE
50 date >> $OUTPUT_FILE

```



# Literaturverzeichnis

- [AMD05] ADVANCED MICRO DEVICES, INC.: *AMD64 Virtualization Codenamed ?Pacifica? Technology; Secure Virtual Machine Architecture Reference Manual*, may 2005. <http://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf>.
- [DME06] DR. MICHAEL ENGEL, MICHAEL HAUPT: *Systemprogrammierung: Virtuelle Maschinen Kapitel 3*, 2006. [http://www.uni-marburg.de/fb12/verteilte\\_systeme/lehre/v1/virtual\\_tech\\_folien/ch03](http://www.uni-marburg.de/fb12/verteilte_systeme/lehre/v1/virtual_tech_folien/ch03).
- [Ent] ENTERPRISES, ALASIR: *The Alasir Licence*. <http://alasir.com/licence/TAL.txt>.
- [Gol73] GOLDBERG, ROBERT P.: *Architectural Principles for Virtual Computer Systems*, Feb 1973. <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=AD772809&Location=U2&doc=GetTRDoc.pdf>.
- [ibm05] *IBM Systems Virtualization Version 2 Release 1*, Dec 2005. <http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf>.
- [itwa] *Paravirtualisierung :: para virtualization :: ITWissen.info*. <http://www.itwissen.info/definition/lexikon/Paravirtualisierung-para-virtualization.html>.
- [itwb] *Virtualisierung :: virtualization technology :: VT :: ITWissen.info*. <http://www.itwissen.info/definition/lexikon/Virtualisierung-VT-virtualization-technology.html>.
- [itwc] *Vollständige Virtualisierung :: full virtualization :: ITWissen.info*. <http://www.itwissen.info/definition/lexikon/Vollstaendige-Virtualisierung-full-virtualization.html>.
- [pcd] PCDREWS: *IOmeter Patch*. [http://sourceforge.net/tracker/index.php?func=detail&aid=1244848&group\\_id=40179&atid=427254](http://sourceforge.net/tracker/index.php?func=detail&aid=1244848&group_id=40179&atid=427254).
- [RMH] RHETT M. HOLLANDER, PAUL V. BOLOTOFF: *RAMspeed, a cache and memory benchmarking tool*. <http://alasir.com/software/ramspeed/>.
- [Spe05] SPECTOR, STEPHEN: *How are Hypervisors Classified?*, Jan 2005. <http://www.xen.org/files/Marketing/HypervisorTypeComparison.pdf>.
- [sysa] *Sync v2.0*. <http://technet.microsoft.com/de-de/sysinternals/bb897438.aspx>.
- [sysb] *Windows Sysinternals*. <http://technet.microsoft.com/de-de/sysinternals/default.aspx>.

## Literaturverzeichnis

- [Van] VANOVER, RICK: *Everyday Virutalization*. <http://virtualizationreview.com/blogs/everyday-virtualization/2009/06/type-1-and-type-2-hypervisors-explained.aspx>.
- [vir] *Paravirtualisierung* :: *para virtualization* :: *ITWissen.info*. <http://www.itwissen.info/definition/lexikon/Paravirtualisierung-para-virtualization.html>.
- [vmwa] *Timekeeping in VMware Virtual Machines*. [http://www.vmware.com/pdf/vmware\\_timekeeping.pdf](http://www.vmware.com/pdf/vmware_timekeeping.pdf).
- [vmwb] *Virtualization History, Virtual Machine, Server Consolidation*. <http://www.vmware.com/virtualization/history.html>.
- [vmwc] *Virtuelle Mschinen, virtueller Server, virtuelle Infrastruktur*. <http://www.vmware.com/de/virtualization/virtual-machine.html>.
- [vmw05] *Virtualization: Architectural Considerations And Other Evalutation Criteria*. 3401 Hillview Ave. Palo Alto CA 94304 USA, 2005. [http://www.vmware.com/pdf/virtualization\\_considerations.pdf](http://www.vmware.com/pdf/virtualization_considerations.pdf).
- [vmw07] *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*. 3401 Hillview Ave. Palo Alto CA 94304 USA, 2007. [http://www.vmware.com/files/pdf/VMware\\_paravirtualization.pdf](http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf).