FHZ → FACHHOCHSCHULE ZENTRALSCHWEIZ

HTA → HOCHSCHULE FÜR TECHNIK+ARCHITEKTUR LUZERN

Abteilung Informatik

Java

Thread Skalierung

STD

Software Test Document

HTA LUZERN
Technikumstrasse 21
CH—6048 Horw

T: 041-349-33-11
F: 041-349-39-60
W: www.hta.fhz.ch

Diplomarbeit 2006
Aregger Marcel
Meier Rainer

Änderungskontrolle

Version	Datum	Ausführende Stelle	Bemerkungen/Art der Änderung
1.1	2006-10-16	Rainer Meier	Initial Release
1.2	2006-11-10	Marcel Aregger	Testcases
1.3	2006-11-14	Marcel Aregger	Resultate Testcases

Prüfung und Freigabe

Vorname/Name	Dokumentversion	Status	Datum	Visum
Rainer Meier	1.3	Final	2006-11-20	
Marcel Aregger	1.3	Final	2006-11-20	

1. Management Summary

Der Startpunkt des vorliegenden Testdokuments bilden die 8 Zielsetzungen im SDD [2] in Bezug auf den geplanten Testumfang in den Ebenen Hardware, Betriebssystem und Java Virtual Machine.

Die effektive Testplattform in Zusammenhang mit der eingesetzten Hardware, bildet eine Fujitsu Siemens Workstation vom Typ Celsius V810 der Firma Pilatus. Dieses SMP-System beinhaltet 2 AMD Opteron 246-Prozessoren mit einer Taktrate von 2GHz. Aufgesetzt auf diese Hardware bilden Windows XP Professional 32 Bit und eine Sun Java HotSpot Client Runtime in der Version 1.5.0_09-b03 die wesentliche Bestandteile der Software Plattform. JOMP-Tests wurden mit einer JOMP-Implementierung in der Version 1.0 Beta durchgeführt. Da einzelne Testcases eine Single-CPU-Architektur erfordern, wurde die oben beschriebene Multi-CPU-Plattform bei Bedarf mit dem Bootflag `numprocs=1` (c:\boot.ini) softwareseitig in eine Single-CPU-Plattform umgewandelt.

Das Testverfahren sichert mit fixen und variablen Rahmenbedingungen die Reproduzierbarkeit und Vergleichbarkeit der Testresultate. Für jeden Testcase sind der Ausschnitt aus der Mandelbrotmenge und die Anzahl Iterationen in der Berechnung dieses Ausschnitts konstante Grössen. Parameter wiebeispielsweise die Anzahl Worker Threads oder die Basispriorität eines Threads werden dem entsprechenden Testcase angepasst. Die Durchführung der Testreihe orientiert sich ebenfalls an spezifische Verhaltensregeln. So wird jede Messung in unveränderter Konfiguration 5-mal wiederholt wobei die Mittelwerte den eigentlichen Messwert bilden. Unrealistische Messwerte oder Programmfehler während der Messung führen zur Wiederholung der ganzen Testreihe.

Insgesamt 10 Testcases bzw. deren Resultate bilden die Grundlage für die Analyse und Interpretation des Systemverhaltens. Sie sind aus den Zielsetzungen {T?} im SDD abgeleitet und können wie folgt zugeordnet werden:

Testcase	#	Zielsetzung Hardware, Betriebssystem und JVM
1	{T1}	Feststellung Grad der Skalierung zw. Single- und Multi-Prozessor-Architektur
2	{T2}	Nachweis Abbildung Java-Thread auf Win32-Thread
3, 4	{T3}	Nachweis Abbildung Java-Thread-Priorität auf Win32-Thread-Priorität
5, 6	{T4}	Analyse Systemverhalten bei Änderung Win32-Thread-Priorität
7	{T5}	Analyse Systemverhalten bei Festlegung einer Prozess-Affinität
8	{T6}	Analyse der Skalierung einer multithreaded Java-Applikation
9	{T7}	Analyse Einfluss der Thread-Synchronisation auf die Skalierung
10	{T8}	Analyse Anwendbarkeit und Effektivität von JOMP

10 Performance Indikatoren wie beispielsweise Berechnungszeit, Kernel Thread Priorität oder die Affinität bilden die Messgrössen der Testcases. Sie werden mit 3 Tools gemessen und protokolliert. Die Konsolenausgabe der Mandelbrot-Anwendung, der CodeAnalyst von AMD und der ProcessExplorer von Sysinternals bilden die Tooling-Plattform.

Alle Tetscases konnten wie geplant abgearbeitet werden und lieferten aussagekräftige Resultate für die nachfolgende Analyse und Interpretation. Sie können in die drei Bereiche Priorität, Affinität und Skalierung gegliedert werden.

Priorität

Im Bereich der Thread-Prioritäten konnte eine lineare Abbildung von Java-Thread- auf Kernel-Thread-Prioritäten nachgewiesen werden. Da jeweils 2 Java-Prioritäten auf eine Kernel-Priorität abgebildet werden, resultieren pro Process Priority Class faktisch 5 nutzbare Prioritätsstufen in Java. Das „extended mapping“, wo zusätzlich die Win32 Process Priority Class mitberücksichtigt wurde, hat gezeigt, dass im Bereich der Basisprioritäten Überlappungen entstehen. Die Kombination verschiedener Java-Thread-Prioritäten und Process Priority Classes ergeben als Resultat die gleiche Basispriorität. Für

die Festlegung der Java Thread Priorität ist somit auch die Priorität des Prozesskontextes im Kernel zu beachten.

Affinität

Für eine multithreaded Java-Anwendung auf Windows XP resultiert eine gleichförmige Verteilung der Threads auf die verfügbaren CPUs des Systems sofern keine andere „Last“ um CPU-Zeit konkurriert. Über die Windows API können Affinitäten auf Level Thread und Prozess definiert werden. In der vorliegenden Versuchsreihe konnte mit dem ProcessExplorer ([PROCEXP]) auf Prozessebene eine Affinität gesetzt werden. Die Zuweisung wurde auf alle Threads dieses Prozesses weitervererbt womit die Kernel-Threads der Java-Anwendung auf eine CPU konzentriert wurden. Versuche unter Einwirkung von konkurrierenden Prozessen haben weiter gezeigt, dass mit dem Setzen einer Affinität der jeweilige Prozessor nicht exklusiv zugeteilt wird.

Skalierung

Ein grundlegender Versuch bestätigte die 1:1-Abbildung eines Java-Threads auf einen Win32-Thread im Kernel. Somit laufen unter Windows XP die Threads einer Java-Anwendung als Kernel-Threads in einem Prozess-Kontext wo sie durch den Scheduler auf verfügbare CPUs verteilt werden können.

Die Skalierung einer Anwendung mit variabler Thread-Anzahl ohne Synchronisation zeigt zwischen der Single- und Multi-Prozessor-Architektur deutliche Unterschiede im Verlauf. Während die 1 CPU-Architektur für 1 bis 512 Threads annähernd konstante Berechnungs- und CPU-Zeiten aufweist, wird auf der 2 CPU-Architektur die Berechnungszeit halbiert. Durch die Verteilung der Threads wird auf der Multi-Prozessor-Umgebung ein Skalierungsfaktor von nahezu 2 erreicht. Gemeinsam haben die Single- und Multi-Prozessor-Architektur, dass der zunehmende Verwaltungsaufwand für 1 bis 512 Threads zu keiner nennenswerten Zunahme der CPU-Zeit führt (ohne Synchronisation).

Das verwendete Synchronisationsverfahren ist für die Skalierung in der Single CPU-Umgebung entscheidend. Mit der Methoden-Synchronisation (grobes Locking) bricht ab >2 Threads die Skalierung um bis zu 50% ein. Wird mit Objekt- oder CAS synchronisiert ergeben sich annähernd konstante Berechnungs- und CPU-Zeiten.

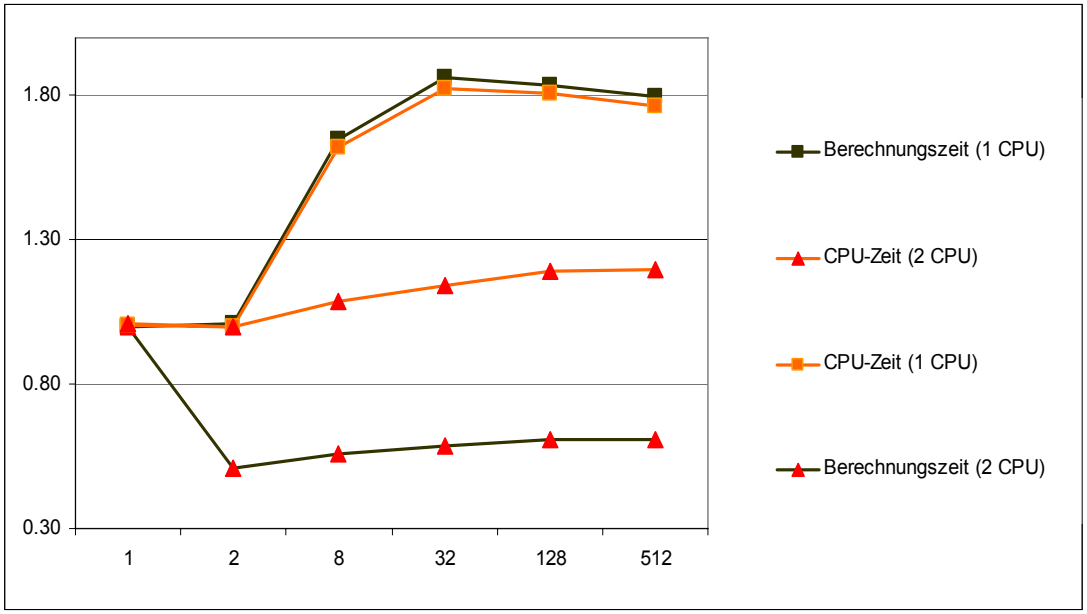

Abbildung 1 Methoden-Synchronisation 1 CPU / 2CPU-Architektur

Im 2 CPU-Umfeld führen verschiedene Synchronisationsverfahren zu einem ähnlichen Verlauf in Bezug auf die benötigte Berechnungs- und CPU-Zeit. Mit 2 Threads wird ein Skalierungsfaktor von annähernd 2 erreicht. Zwischen 3 bis 512 Threads ergibt sich dann abhängig von der gewählten Synchronisierungsvariante eine Zunahme der Berechnungs- und CPU-Zeit die zwischen 4% und 20% liegt. Ein 2 CPU-System wird also durch die Verwendung einer Methoden-Synchronisation weit weniger ausgebremst als ein 1 CPU-System.

2. Inhaltsverzeichnis

1. Management Summary 3
2. Inhaltsverzeichnis 5
3. Dokumentinformationen 8
3.1. Referenzierte Dokumente 8
3.2. Definitionen und Abkürzungen 8
3.3. Links 9
4. Einleitung 10
4.1. Zweck des Dokuments 10
5. Test-Scope 11
6. Test-Plattform 12
6.1. Hardware-Dokumentation 12
6.2. Software-Dokumentation 13
7. Testverfahren 15
7.1. Rahmenbedingungen 15
7.1.1. Fixe Grössen 15
7.1.2. Variable Gössen 16
7.2. Durchführung der Testreihe 16
7.3. Anforderungen Testcases 17
8. Testcases 18
8.1. Testcase Hardware 18
8.1.1. Testcase 1 18
8.2. Testcases Betriebssystem 19
8.2.1. Testcase 2 19
8.2.2. Testcase 3 19
8.2.3. Testcase 4 20
8.2.4. Testcase 5 21
8.2.5. Testcase 6 22
8.2.6. Testcase 7 23
8.3. Testcases JVM 24
8.3.1. Testcase 8 24
8.3.2. Testcase 9 25
8.3.3. Testcase 10 26
9. Messen und Protokollieren 27
9.1. Performance Indikatoren 27
9.2. Profiling und Testtools 28
9.2.1. Übersicht der Tools 28
9.2.2. Ausgabe Testtools 29

10. Ergebnisse der Testcases 31
10.1. Testcase 1 31
10.1.1. Berechnungszeit 31
10.1.2. CPU-Zeit 32
10.1.3. Ergebnisse 32
10.2. Testcase 2 33
10.2.1. Thread Mapping 33
10.2.2. Ergebnisse 33
10.3. Testcase 3 34
10.3.1. Priority-Mapping 34
10.3.2. Ergebnisse 34
10.4. Testcase 4 35
10.4.1. Priority-Mapping 35
10.4.2. Ergebnisse 36
10.5. Testcase 5 37
10.5.1. Berechnungszeit 37
10.5.2. CPU-Zeit 38
10.5.3. Ergebnisse 38
10.6. Testcase 6 39
10.6.1. Berechnungszeit (variable Win32-Priorität) 39
10.6.2. Berechnungszeit (variable Laststufe) 40
10.6.3. CPU-Zeit 41
10.6.4. Ergebnisse 41
10.7. Testcase 7 42
10.7.1. Ergebnisse 43
10.8. Testcase 8 45
10.8.1. Skalierung 1 CPU ohne Synchronisation 45
10.8.2. Skalierung 2 CPU ohne Synchronisation 46
10.8.3. Ergebnisse 46
10.9. Testcase 9 48
10.9.1. Skalierung 1 CPU - Methodensynchronisation 48
10.9.2. Skalierung 1 CPU - Objektsynchronisation 49
10.9.3. Skalierung 1 CPU – CAS-Synchronisation 50
10.9.4. Ergebnisse 51
10.9.5. Skalierung 2 CPU - Methodensynchronisation 52
10.9.6. Skalierung 2 CPU - Objektsynchronisation 53
10.9.7. Skalierung 2 CPU - CAS-Synchronisation 54
10.9.8. Ergebnisse 55
10.10. Testcase 10 57
10.10.1. Skalierung 2 CPU - JOMP-Threads 57

10.10.2. Ergebnisse 58
11. Glossar 59
12. Verzeichnisse 60
12.1. Tabellenverzeichnis 60
12.2. Abbildungsverzeichnis 61
12.3. Code Listings 61
12.4. Index 61

3. Dokumentinformationen

3.1. Referenzierte Dokumente

Tabelle 1 Referenzierte Dokumente

Referenz	Beschreibung
[1]	Basisanalyse
[2]	SDD, Software Design Document
[3]	Systeminformationen „System-Information.nfo“
[4]	JavaDev JumpStart, Java Entwicklungsumgebung
[5]	Conclusion, Schlussfolgerung Projektergebnisse

3.2. Definitionen und Abkürzungen

Tabelle 2 Abkürzungen

Abkürzung	Beschreibung
API	Application Programming Interface
CPU	Central Processing Unit
DEP	Date Execution Prevention (siehe auch NX)
HAT	HyperTransport
JVM	Java Virtual Machine
NX	No eXecute
SDD	Software Design Document
STD	Software Test Document

3.3. Links

Tabelle 3 Links

Referenz	Beschreibung
[BOOTFLAG]	Windows XP boot parameters: http://support.microsoft.com/kb/833721
[SUNJAVA]	Sun Java Home: http://java.sun.com/
[ECLIPSE]	Eclipse IDE: http://www.eclipse.org/
[APACHEANT]	Apache ANT, Java Builder: http://ant.apache.org/
[JOMP]	EPCC, OpenMP-like directives for Java: http://www.epcc.ed.ac.uk/research/jomp/
[CODEANALYST]	AMD CodeAnalyst: http://developer.amd.com/cawin.jsp
[PROCEXP]	Sysinternals Process Explorer: http://www.microsoft.com/technet/sysinternals/utilities/ProcessExplorer.mspx

4. Einleitung

4.1. Zweck des Dokuments

Das Software Test Document STD beinhaltet alle notwendigen Elemente für die konkrete Umsetzung des geplanten Testumfanges im SDD ([2]). Es definiert dabei die effektiv eingesetzten Hardware- und Softwarekomponenten und beschreibt mit dem Testverfahren die Rahmenbedingungen und Verhaltensregeln für die Durchführung einer transparenten Testreihe.

Das eigentliche Testing wird in Form von Testcases beschrieben und erstreckt sich im Sinne der layerorientierten Betrachtung über die Layer Hardware, Betriebssystem und JVM. Das Systemverhalten in den einzelnen Testcases wird über vorab definierte Performance Indikatoren (Messgrössen) gemessen. Testcases und deren Performance Indikatoren sind ebenfalls Teil des STDs.

Die Messung und Protokollierung der oben genannten Indikatoren muss über spezifische Instrumente (Tools) erfolgen, die in diesem Dokument festgelegt werden. Sie ermöglichen reproduzierbare Ergebnisse, die nachfolgen verifiziert werden können.

Ergebnisse aus den Testcases werden in komprimierter Form in diesem Dokument dargestellt. Die Interpretation des Systemverhaltens ist aber Teil des Conclusion ([5]).

5. Test-Scope

Der geplante Testumfang im Software Design Document SDD (Kapitel 6) bietet einen ersten Überblick über jene Aspekte, die im Zusammenhang mit der Skalierung einer Applikation analysiert und getestet werden müssen. Für diese Aspekte, die in Form von Zielsetzungen und zugehörigen Betrachtungsbereichen beschrieben wurden, werden im vorliegenden Software Test Document STD entsprechende Testverfahren und Testcases abgeleitet. Der Testumfang im SDD mit den insgesamt 8 Hauptzielsetzungen {T1}...{T8} (siehe Tabelle 4) ist in diesem Zusammenhang als Guideline zu betrachten.

Tabelle 4 Geplanter Test-Umfang gemäss SDD ([2])

#	Zielsetzung	Kategorie
{T1}	Feststellung Grad der Skalierung zwischen Single- und Multi-Prozessor-Architektur	Muss
{T2}	Nachweis Abbildung Java-Thread auf Win32-Thread	Muss
{T3}	Nachweis Abbildung Java-Thread-Priorität auf Win32-Thread-Priorität	Kann
{T4}	Analyse Systemverhalten bei Änderung Win32-Thread-Priorität	Kann
{T5}	Analyse Systemverhalten bei Festlegung einer Prozess-Affinität	Kann
{T6}	Analyse der Skalierung einer multithreaded Java-Applikation	Muss
{T7}	Analyse Einfluss der Thread-Synchronisation auf die Skalierung	Kann
{T8}	Analyse Anwendbarkeit und Effektivität von JOMP	Muss

6. Test-Plattform

Da uns die Schule keine geeignete Testplattform (Multi-Core/Multi/CPU, Windows XP) zur Verfügung stellen konnte haben wir diese selber beschafft. Unser Dank geht hier an die Pilatus Flugzeugwerke AG in Stans die uns freundlicherweise die Hardware für die Dauer der Diplomarbeit zur Verfügung gestellt hat.

Die verwendete Hard- und Software wird nachfolgend möglichst detailliert erfasst. Zusätzlich wird ein Microsoft System Information 7 Dokument (siehe [3]) abgegeben. Dieses enthält alle wichtigen Eckdaten des Systems um die Reproduzierbarkeit zu gewährleisten.

6.1. Hardware-Dokumentation

Abbildung 2 Hardware Testplattform

Alle Tests wurden auf einer Fujitsu Siemens Celsius V810 durchgeführt. Die folgende Tabelle beinhaltet die wichtigsten Eckdaten des verwendeten Systems:

Tabelle 5 Hardware-Eckdaten

Bezeichnung	Konfiguration
Hersteller	Fujitsu Siemens
Modell	Celsius V810
Anzahl Prozessoren	2
Prozessor Typ	AMD Opteron 246 (Sledgehammer Core), 0.13µ SOI, Version 2.0, Generation 15, Model 5, Stepping 8, Single-Core, NX-Technology, Socket 940
Cache	L1: 64kB Instruction Cache, 64kB Data-Cache L2: 1MB
Taktrate	2GHz
HT-Speed	800MHz
Speicher	4GB DDR333 (166MHz) non-ECC, 184 Pin
Chipset	AMD 8131
Festplatte(n)	2x 80GB RAID-0 Array (160GB Total), 1 Primäre NTFS Partition
DVD	1x HL-DT-SR RW/DVD GCC-4480B
Netzwerk	Broadcom NetXtreme Gigabit Ethernet
Grafik	nVidia GeForce Quadro FX 1100, 128MB, Driver Version 6.14.10.7756

6.2. Software-Dokumentation

Für die Tests wurden absichtlich keine Windows-Dienste deaktiviert, Registry-Optimierungen oder ähnliches vorgenommen. Beim Betriebssystem handelt es sich somit um ein standardmässig installiertes System. Dies sollte somit weitgehend der üblichen Konfiguration einer Workstation entsprechen womit auch die Ergebnisse auf ähnlichen Maschinen und im Praxiseinsatz vergleichbar sind.

Die relevante, zusätzlich installierte Software ist in folgender Tabelle festgehalten:

Tabelle 6 Software-Umgebung

Bezeichnung	Beschreibung
Windows XP Professional 32 Bit	Als Betriebssystem kommt Windows XP Professional in der 32 Bit Version mit ServicePack 2 zum Einsatz.
Java VM	Wir verwenden für alle Tests die Sun Java HotSpot Client Runtime in der Version `build 1.5.0_09-b03`. Weitere Informationen unter [SUNJAVA].
Eclipse	Zur Entwicklung wird Eclipse in der Version 3.2.1 verwendet. Zur Unterstützung verwenden wir einige Plugins wie den Visual Editor für die Erzeugung der GUI-Klassen. Weitere Informationen unter [ECLIPSE].
Apache-ANT	Der Build-Support wird mit Hilfe von Apache-ANT realisiert. Dies gewährleistet die Plattformunabhängigkeit und erlaubt die Kompilierung auch ohne Eclipse. Weitere Informationen unter [APACHEANT].
Java-Dev JumpStart	Sowohl die Sun Java VM als auch Eclipse incl. Plugins und weitere Hilfsprogramme wie Apache ANT sind Teil dieses Paketes. Das Installationsprogramm wird zusammen mit dieser Arbeit abgegeben und erlaubt die Installation der gesamten Java-Umgebung innerhalb weniger Minuten. Siehe auch [4].
JOMP	Für die JOMP Implementierung wird JOMP in der Version 1.0 Beta verwendet; die zu diesem Zeitpunkt aktuellste Version. Siehe auch [PROCEXP].

Die folgende Tabelle beinhaltet eine Auflistung der durchgeführten Konfigurationsanpassungen:

Tabelle 7 Software Konfigurationsanpassung

Bezeichnung	Beschreibung
/PAE Kernel Parameter	Der /PAE Parameter (Physical Address Translation) wird häufig wegen der Speicherunterstützung über 4GB verwendet. Dies ist für unsere Tests zwar nicht relevant aber aktiviert auch die Hardware-DEP (Date Execution Prevention) Unterstützung. Da unser Opteron basierendes System das NX Bit (gleichzusetzen mit der Microsoft-Bezeichnung DEP) unterstützt haben wir es auch aktiviert. Der Parameter wird in c:\boot.ini eingetragen. Siehe auch [BOOTFLAG].
/numprocs=1	Für unsere Single-CPU Messungen haben wir den zusätzlichen Parameter /numprocs=1 verwendet. Dieser teilt dem Kernel die maximale Anzahl zu verwendender Prozessoren mit. Dies erlaubt uns die Verwendung desselben Systems für Single- und Multiprozessor Tests. Der Parameter wird in c:\boot.ini eingetragen und nur Singleprozessor Tests verwendet. Siehe auch [BOOTFLAG].

Wie bereits erwähnt liegt der Arbeit eine Ausgabe von Microsoft System Information 7 bei (siehe [3]). Das Programm kann über „Start ➔ Ausführen ➔ msinfo32“ gestartet werden. Um die Lesbarkeit zu garantieren haben wir ausserdem noch einen Textbasierenden Export beigelegt.

7. Testverfahren

Das Testverfahren legt die Rahmenbedingen und Verhaltensregeln der geplanten Testreihe fest. Weiter werden in diesem Kapitel die formellen Anforderungen an die Testcases beschrieben. Die nachfolgenden Themenbereiche dienen dazu, die Testreihe effizient zu gestalten und die Nachvollziehbarkeit der Resultate sicherzustellen.

7.1. Rahmenbedingungen

Rahmenbedingungen sind als grundlegende Voraussetzung für die Durchführung eines Tests zu betrachtet. Sie können aus der Aufgabenstellungen abgeleitet werden (beispielsweise beim Betriebssystem: Windows XP) oder ergeben sich aus dem jeweiligen Testcase (beispielsweise ein Tool für variable CPU-Belastung; Windows Calculator). Die Rahmenbedingungen können differenziert werden in Bedingungen, die global für alle Testcases gelten und jene, die spezifisch für einzelne Testcases anzuwenden sind.

7.1.1. Fixe Grössen

Fixe Rahmenbedingungen gelten für alle Testcases gleichermassen und werden darum in den Testcases nicht explizit aufgeführt:

Ausschnitt Mandelbrotmenge

Um die Resultate der Benchmarks vergleichbar zu machen, wird für jeden Testcase und Berechnungsdurchlauf der gleiche Ausschnitt aus der Mandelbrot-Menge berechnet. Dazu müssen die exakten Koordinaten dieses Ausschnitts persistent gespeichert werden können. Eine Bookmark-Datei mit der Definition des entsprechenden Bereiches wird unter dem Namen `benchmark-location.xml` mitgeliefert.

Anzahl Iterationen

Mit der Anzahl Iterationen kann der Berechnungsaufwand der Mandelbrot-Menge beliebig gesteuert werden. Gemäss Requirement {R1.3} (SDD, Kapitel 7.1) soll die Laufzeit (Berechnungszeit) des Programms zwischen 30 Sekunden und 5 Minuten betragen. Für die Durchführung der Testreihe soll eine geeignete „Iterationstiefe“ ermittelt und generell angewendet werden. Eine Bookmark-Datei mit der Definition der entsprechenden Iterationstiefe wird unter dem Namen `benchmark-location.xml` mitgeliefert.

Bildgrösse

Der Berechnungsaufwand hängt natürlich auch von der Anzahl der berechneten Bildpunkte ab. Um diese konstant zu halten wird das Hauptfenster in der Initialgrösse von 1024x768 Pixel belassen. Dies resultiert in einer darstellbaren Bildgrösse von 1016x718 Pixel oder umgerechnet 729'488 Bildpunkte.

7.1.2. Variable Gössen

Variable Rahmenbedingungen sind für den jeweiligen Testcase spezifisch festzulegen und somit in der Beschreibung dieses Testcases aufzuführen.

Anzahl CPU

Für plattformübergreifende Testcases (1 CPU / 2 CPU) ist die Anzahl verfügbarer CPUs festzulegen. Sind diesbezüglich keine Einschränkungen erwähnt, werden Tests auf der 2 CPU-Architektur ausgeführt.

Anzahl Worker Threads

Die Pixel der Mandelbrot-Menge werden von 1...n Worker Threads berechnet. Da diese Worker auf Kernel-Threads abgebildet und somit verteilt werden können, ist deren Anzahl im voraus festzulegen. Die Menge der Threads, die an der Berechnung teilnehmen, ist für einen Testcase konstant oder entsprechend der Testreihe anzupassen.

Priorität eines Prozess/Threads

Die Basispriorität eines Threads beeinflusst das Schedulingverhalten des Kernel und somit die Laufzeit des Programms. Die festgelegte Basispriorität der Threads eines Prozesses ist für einen Testcase konstant oder entsprechend der Testreihe anzupassen.

Synchronisation/Locking

Der Verwaltungsaufwand im Zusammenhang mit der Synchronisation von Threads kann die Berechnungs- und CPU-Zeit direkt beeinflussen. Die Anwendung von Synchronisation bzw. Synchronisations-Mechanismus ist für einen entsprechenden Testcase auszuweisen.

7.2. Durchführung der Testreihe

Der Ablauf einer Testreihe ist auf Level Testcase festgelegt und beinhaltet folgende grundsätzlichen Verhaltensregeln:

- Eine spezifische Messung wird mit der unveränderten Konfiguration 5 mal durchgeführt.
- Sind Abweichungen der Teilresultate vernachlässigbar, kann die Durchlaufzahl auf mind. 3 reduziert werden.
- Die Teilresultate der Messungen werden in Excel erfasst.
- Das Mittel der Teilresultate bildet den eigentlichen Messwert.
- Ergibt eine Messung ein unrealistisches Teilresultat, wird der ganze Durchgang wiederholt.
- Tritt während einer Messung ein Programmfehler auf, wird der ganze Durchgang wiederholt.
- Tritt der Programmfehler erneut auf, wird dieser Test übersprungen.
- Offene Tests werden nach der Korrektur des Programms nachgeholt.
- Auswirkungen der Programmänderung auf bereits durchgeführte Tests sind zu analysieren.
- Nach jedem Testcase werden die Applikation neu gestartet (Mandelbrot, Systools).
- Single-CPU Tests werden mit gesetztem Kernel-Flag `/numproc=1` in c:\boot.ini getestet.

7.3. Anforderungen Testcases

Um die Tests effektiv und effizient durchführen zu können, müssen die Testcases bestimmte formelle Bedingungen erfüllen. Als minimale Anforderung müssen folgende Punkte für jeden Testcase beschrieben werden:

- ID und zugehörige Zielsetzung aus dem SDD ({T?})
- Erwartete(s) Resultat(e)
- (Variable) Rahmenbedingen
- Performance Indikatoren (Messgrössen)
- Spezifische Ablaufschritte

8. Testcases

Die nachfolgenden Testcases realisieren den geplanten Testumfang aus dem SDD ([2]). Sie beschreiben für die Layer Hardware, Betriebssystem und JVM alle Testszenarien die durchgeführt werden müssen um die formulierten Zielsetzungen umzusetzen. Die Ergebnisse aus diesen Testcases sind im Kapitel 10; Ergebnisse der Testcases einsehbar.

8.1. Testcase Hardware

8.1.1. Testcase 1

Testcase für Zielsetzung {T1} aus geplantem Test-Umfang SDD ([2])

Testcase 1	**Betrachtungsbereich: Hardware**
Zielsetzung	Feststellung Grad der Skalierung zwischen Single- und Multi-Prozessor-Architektur
Erwartetes Resultat	Faktor über potenzielle (vertikale) Skalierung zwischen Single- und Multi-Prozessor-Architektur einer Java-Anwendung ohne Berücksichtigung von Einflüssen wie bspw. Synchronisation und Prioritäten
Rahmenbedingungen	Vergleichbare Testplattform mit Single- und Multi-Prozessor
	Single-Threaded Testklasse erweiterbar auf Multi-Threaded
	Messbare Berechnungszeit und Ressourcenbedarf
Performance-Indikatoren	Verfügbare CPU
	Aktive Threads
	Berechnungszeit
	CPU-Zeit
	Kontextwechsel
Ablaufschritte	Anwendung; Testklasse auf 1 CPU-Architektur
	Anwendung; Anzahl Worker-Threads (1 | 2) festlegen
	Protokollieren Performance-Indikatoren
	Anwendung; Testklasse auf 2 CPU-Architektur
	Anwendung; Anzahl Worker-Threads (1 | 2) festlegen
	Protokollieren Performance-Indikatoren

8.2. Testcases Betriebssystem

8.2.1. Testcase 2

Testcase für Zielsetzung {T2} aus geplantem Test-Umfang SDD ([2])

Testcase 2	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Nachweis Abbildung Java-Thread auf Win32-Thread
Erwartetes Resultat	Beweis für 1:1-Abbildung von Java-Thread auf Win32-Thread.
	Identifikation Deamon-Threads (Anzahl) für spezifische Java-Anwendung
	Identifikation zugehörigen Prozesskontext auf Level Betriebssystem
Rahmenbedingungen	Betriebssystem Windows XP
	VM mit Native-Thread-Unterstützung
	Java-Anwendung mit variabler Thread-Anzahl
Performance-Indikatoren	Java-Threads
	Kernel-Threads
	Prozesskontext Kernel-Threads
Ablaufschritte	Device; Processexplorer starten
	Anwendung; Mandelbrot starten
	Anwendung; Threadanzahl (n) in Konfigmenü festlegen
	Systemtool; Resultierende Threadanzahl in Processexplorer ermitteln
	Testablauf mit Threadanzahl (!= n) wiederholen
	Anzahl Deamon-Threads bestimmen

8.2.2. Testcase 3

Testcase für Zielsetzung {T3} aus geplantem Test-Umfang SDD ([2])

Testcase 3	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Nachweis Abbildung Java-Thread-Priorität auf Win32-Thread-Priorität mit unveränderter Default Process Priority-Class
Erwartetes Resultat	Mapping Tabelle die Java-Thread Priorität (1...5...10) auf die Kernel-Thread-Priorität abbildet (Basispriorität) unter Verwendung von Default-Einstellung für Process Priority-Class
Rahmenbedingungen	Betriebssystem Windows XP
	Java-Anwendung mit veränderbarer Thread-Priorität (Konfigmenü)
Performance-Indikatoren	Java Thread Priorität

	Kernel Thread Priorität
Ablaufschritte	Systemtool; Processexplorer starten
	Anwendung; Mandelbrot starten
	Anwendung; Java Thread Priorität(n) in Konfigmenü festlegen
	Systemtool; Resultierende Kernel Thread Priorität in Processexplorer ermitteln
	Test mit ; Java Thread Priorität (!= n) wiederholen
	Priorität-Mapping durchführen, Java Thread Priorität (n= 1...5...10)

8.2.3. Testcase 4

Testcase für Zielsetzung {T3} aus geplantem Test-Umfang SDD ([2])

Testcase 4	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Nachweis Abbildung Java-Thread-Priorität auf Win32-Thread-Priorität mit variabler Process Priority-Class
Erwartetes Resultat	Mapping Tabelle die Java-Thread Priorität (1...5...10) auf die Kernel-Thread-Priorität abbildet (Basispriorität) unter Verwendung möglicher Einstellungen für die Process Priority-Class
Rahmenbedingungen	Betriebssystem Windows XP
	Java-Anwendung mit veränderbarer Thread-Priorität (Konfigmenü)
	Systemtool mit dem Process Priority Class eigestellt werden kann
Performance-Indikatoren	Java Thread Priorität
	Kernel Thread Priorität
Ablaufschritte	Systemtool; Processexplorer starten
	Anwendung; Mandelbrot starten
	Anwendung; Java Thread Priorität (n= 1...5...10) in Konfigmenü festlegen
	Systemtool; Process Priority Class festlegen
	Systemtool; Resultierende Kernel Thread Priorität in Processexplorer ermitteln
	Test mit Java Thread Priorität (n= 1...5...10) und allen Process Priority Classes wiederholen
	Priorität-Mapping durchführen Java Thread Priorität, Process Priority Class, Kernel Thread Priorität

8.2.4. Testcase 5

Testcase für Zielsetzung {T4} aus geplantem Test-Umfang SDD ([2])

Testcase 5	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Analyse Systemverhalten (Skalierung) bei Änderung der Process Priority Class und unterschiedlicher Last
Erwartetes Resultat	Faktor der Skalierung auf Multi-CPU-Maschine für Anwendungen mit gleicher Kernel Thread Priorität bei unterschiedlicher Process Priority Classes unter Berücksichtigung variabler Laststufen
Rahmenbedingungen	Betriebssystem Windows XP
	Priorität-Mapping Java Thread Priorität, Process Priority Class und Kernel Thread Priorität
	Java-Anwendung mit veränderbarer Thread-Priorität (Konfigmenü)
	Systemtool mit dem Process Priority Class eigestellt werden kann
	Tool für variable CPU-Belastung (Windows Calculator, 999999!)
Performance-Indikatoren	Kernel Thread Priorität
	Process Priority Class
	Berechnungszeit
	CPU-Zeit
Ablaufschritte	Systemtool; Processexplorer starten
	Anwendung; Mandelbrot starten
	Anwendung; Java Thread Priorität (m) in Konfigmenü so festlegen dass mit spezifischer Process Priority Class (n) die gewünschte Kernel Thread Priorität (p) resultiert
	Systemtool; Process Priority Class (n) mit Systemtool so festlegen dass mit spezifischer Java Thread Priorität (m) die gewünschte Kernel Thread Priorität (p) resultiert
	Calculator; CPU-Belastung festlegen (keine | mittel | stark)
	Anwendung; Mandelbrotberechnung starten
	Protokollieren Performance-Indikatoren
	Test mit unveränderter Einstellung der Prioritäten (m, n, p) aber veränderter Lastsituation wiederholen
	Test mit gleicher Kernel Thread Priorität (p) aber unterschiedlicher Process Priority Class (n') wiederholen
	Test mit unveränderter Einstellung der Prioritäten (m', n', p) aber veränderter Lastsituation wiederholen

8.2.5. Testcase 6

Testcase für Zielsetzung {T4} aus geplantem Test-Umfang SDD ([2])

Testcase 6	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Analyse Systemverhalten (Skalierung) bei Änderung Win32-Thread-Priorität und unterschiedlicher Last
Erwartetes Resultat	Faktor der Skalierung auf Multi-CPU-Maschine für Anwendungen mit unterschiedlicher Kernel Thread Priorität und Process Priority Classes unter Berücksichtigung variabler Laststufen
Rahmenbedingungen	Betriebssystem Windows XP
	Priorität-Mapping Java Thread Priorität, Process Priority Class und Kernel Thread Priorität
	Java-Anwendung mit veränderbarer Thread-Priorität (Konfigmenü)
	Systemtool mit dem Process Priority Class eigestellt werden kann
	Tool für variable CPU-Belastung (Windows Calculator, 999999!)
Performance-Indikatoren	Kernel Thread Priorität
	Process Priority Class
	Berechnungszeit
	CPU-Zeit
Ablaufschritte	Systemtool; Processexplorer starten
	Anwendung; Mandelbrot starten
	Anwendung; Java Thread Priorität (m) in Konfigmenü so festlegen dass mit spezifischer Process Priority Class (n) die gewünschte Kernel Thread Priorität (p) resultiert
	Systemtool; Process Priority Class (n) mit Systemtool so festlegen dass mit spezifischer Java Thread Priorität (m) die gewünschte Kernel Thread Priorität (p) resultiert
	Calculator; CPU-Belastung festlegen (keine | mittel | stark)
	Anwendung; Mandelbrotberechnung starten
	Protokollieren Performance-Indikatoren
	Test mit unveränderter Einstellung der Prioritäten (m, n, p) aber veränderter Lastsituation wiederholen
	Test mit ungleicher Kernel Thread Priorität (p') und unterschiedlicher Process Priority Class (n') wiederholen
	Test mit unveränderter Einstellung der Prioritäten (m', n', p') aber veränderter Lastsituation wiederholen

8.2.6. Testcase 7

Testcase für Zielsetzung {T5} aus geplantem Test-Umfang SDD ([2])

Testcase 7	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Analyse Systemverhalten bei Festlegung einer Prozess-Affinität
Erwartetes Resultat	Nachweis Prozess-Affinität wird auf Kernel Threads dieses Prozesses vererbt
	Nachweis Auslastung kann mit Affinität auf 1 Prozessor konzentriert werden
	Nachweis über Systemtool / Affinität kann kein „Ideal Processor“ gesetzt werden
	Nachweis Threads eines Prozesses mit Affinität auf eine CPU konkurrieren auf dieser CPU mit Threads ohne explizite Affinität
Rahmenbedingungen	Betriebssystem Windows XP
	Systemtool mit dem Affinität gesetzt werden kann
	2 CPU-Maschine
	Tool für variable CPU-Belastung (Windows Calculator, 999999!)
Performance-Indikatoren	Kernel Thread Priorität
	Affinität
	Berechnungszeit
Ablaufschritte	Systemtool; Processexplorer starten
	Anwendung; Mandelbrot starten
	Anwendung; Mandelbrotberechnung starten ohne Prozess-Affinität
	Protokollieren Performance-Indikatoren
	Systemtool; Anwendungs-Prozess selektieren und Affinität setzen (CPU 1)
	Anwendung; Mandelbrotberechnung starten mit Prozess-Affinität (CPU 1)
	Protokollieren Performance-Indikatoren
	Calculator; CPU-Belastung festlegen, ohne Calc-Prozess Affinität
	Anwendung; Mandelbrotberechnung starten mit Prozess-Affinität (CPU 1)
	Protokollieren Performance-Indikatoren
	Calculator; CPU-Belastung festlegen, mit Calc-Prozess Affinität (CPU 0)
	Anwendung; Mandelbrotberechnung starten mit Prozess-Affinität (CPU 1)
	Protokollieren Performance-Indikatoren

8.3. Testcases JVM

8.3.1. Testcase 8

Testcase für Zielsetzung {T6} aus geplantem Test-Umfang SDD ([2])

Testcase 8	**Betrachtungsbereich: JVM**
Zielsetzung	Analyse der Skalierung einer multithreaded Java-Applikation
Erwartetes Resultat	Faktoren für Skalierung zwischen Single- und Multi-Prozessor-Architektur einer Java-Anwendung mit 1...n Threads. Die Analyse erfolgt unter Berücksichtigung variabler Java Thread Anzahl und ohne Berücksichtigung der Thread Synchronisation
	Charakteristischer Verlauf der Performance-Indikatoren auf Single- und Multi-Prozessor-Architektur mit 1...n Threads und variablen Thread Prioritäten
Rahmenbedingungen	Vergleichbare Testplattform mit Single- und Multi-Prozessor
	Single-Threaded Testklasse erweiterbar auf Multi-Threaded
	Java-Anwendung mit veränderbarer Thread-Priorität (Konfigmenü)
	Messbare Berechnungszeit und Ressourcenbedarf
Performance-Indikatoren	Verfügbare CPU
	Aktive Threads
	Java Thread Priorität
	Berechnungszeit
	CPU-Zeit
Ablaufschritte	Anwendung; Testklasse auf 1 CPU-Architektur
	Systemtool; Processexplorer starten
	Anwendung; Anzahl Worker-Threads (1) festlegen
	Anwendung; Mandelbrot starten
	Anwendung; Java Thread Priorität(1) in Konfigmenü festlegen
	Anwendung; Mandelbrotberechnung starten
	Protokollieren Performance-Indikatoren
	Testablauf wiederholen für (2...n) Threads und Java Thread Prio (2...5...10)
	Anwendung; Testklasse auf 2 CPU-Architektur
	Testablauf analog 1 CPU-Architektur

8.3.2. Testcase 9

Testcase für Zielsetzung {T7} aus geplantem Test-Umfang SDD ([2])

Testcase 9	**Betrachtungsbereich: JVM**
Zielsetzung	Analyse Einfluss der Thread-Synchronisation auf die Skalierung
Erwartetes Resultat	Faktoren für Skalierung zwischen Single- und Multi-Prozessor-Architektur einer Java-Anwendung mit 1...n Threads. Die Analyse erfolgt unter Berücksichtigung variabler Java Thread Anzahl und Thread Synchronisation
	Charakteristischer Verlauf der Performance-Indikatoren auf Single- und Multi-Prozessor-Architektur mit 1...n Threads, variablen Thread Prioritäten und Synchronisation
Rahmenbedingungen	Vergleichbare Testplattform mit Single- und Multi-Prozessor
	Single-Threaded Testklasse erweiterbar auf Multi-Threaded
	Java-Anwendung mit veränderbarer Thread-Priorität (Konfigmenü)
	Java-Anwendung mit Synchronisation (mittlerer/grosser lock contention)
	Messbare Berechnungszeit und Ressourcenbedarf
Performance-Indikatoren	Verfügbare CPU
	Aktive Threads
	Java Thread Priorität
	Berechnungszeit
	CPU-Zeit
Ablaufschritte	Anwendung; Testklasse auf 1 CPU-Architektur
	Systemtool; Processexplorer starten
	Anwendung; Anzahl Worker-Threads (1) festlegen
	Anwendung; Mandelbrot starten
	Anwendung; Java Thread Priorität(1) in Konfigmenü festlegen
	Anwendung; Mandelbrotberechnung starten
	Protokollieren Performance-Indikatoren
	Testablauf wiederholen für (2...n) Threads und Java Thread Prio (2...5...10)
	Anwendung; Testklasse auf 2 CPU-Architektur
	Testablauf analog 1 CPU-Architektur

8.3.3. Testcase 10

Testcase für Zielsetzung {T8} aus geplantem Test-Umfang SDD ([2])

Testcase 10	**Betrachtungsbereich: JVM**
Zielsetzung	Parallelisierung durch JOMP
Erwartetes Resultat	Automatische Threadbildung durch JOMP-Direktiven in Abhängigkeit der Konfiguration (`setNumThreads()`)
Rahmenbedingungen	Testklasse mit for-Schleife die in JOMP-Direktiven gekapselt werden kann
	JOMP-Precompiler
Performance-Indikatoren	Verfügbare CPU
	Java-Threads (JOMP)
	Berechnungszeit
	CPU-Zeit
Ablaufschritte	Anwendung; Testklasse auf 2 CPU-Architektur
	Pre-Compiler; Testklasse(n) kompilieren
	Anwendung; JOMP konfigurieren (Anzahl Threads festlegen)
	Anwendung; JOMP Testlauf starten
	Protokollieren Performance-Indikatoren
	Testablauf mit veränderter Konfiguration (Thread Anzahl) widerholen

9. Messen und Protokollieren

Die Skalierung bzw. das Systemverhalten soll auf Basis von Performance Indikatoren analysiert und dokumentiert werden. Jeder Testcase definiert „seine“ spezifischen Indikatoren die in Tabelle 1 zusammengefasst sind. Für diese relevanten Faktoren werden unter 9.2; Profiling und Testtools entsprechende Instrumente definiert mit denen die Messung und Protokollierung durchgeführt werden kann.

9.1. Performance Indikatoren

Tabelle 8 Relevante Performance Indikatoren

Performance-Indikator	Beschreibung
Verfügbare CPU	Anzahl CPU (Cores) die Rechenzeit zu Verfügung stellen
Java-Thread	Anzahl instanzierter Worker-Threads die Berechnung durchführen
Kernel-Thread	Anzahl erzeugter Kernel-Threads aus Java-Threads
Java Thread Priorität	Aktuelle Priorität des Worker-Threads
Kernel Thread Priorität	Aktuelle Priorität des Kernel-Level-Threads (Basispriorität)
Process Priority Class	Priorität des Win32-Prozesses
Kontextwechsel	Totale Anzahl Kontextwechsel alle Threads
Affinität	Explizite Zuordnung eines Prozesses auf eine CPU
Berechnungszeit	Zeitpunkt vom Start bis zum Ende einer Berechnung (Gesamtbild)
CPU-Zeit	Aufkumulierte CPU-Zeit des Java Prozesses

Die Laufzeit (Prozesslebensdauer) der Java-Anwendung ist von der eigentlichen Berechnungszeit abzugrenzen. Sie beinhaltet neben der/den Berechnungszeit(en) auch die Zeit für das Setup der Anwendung (Erzeugung und Konfiguration der Threads) und ist darum für Performance-Messungen irrelevant.

Tabelle 9 Nicht relevante Performance Indikatoren

Performance-Indikator	Beschreibung
Laufzeit	Zeitpunkt vom Start bis zum Ende einer Anwendung (Prozesslebensdauer)

9.2. Profiling und Testtools

9.2.1. Übersicht der Tools

Tabelle 10 Profiling- und Testtools

Performance-Indikator	Profiling	Protokollierung
Berechnungszeit	Code (Mandelbrot-Anwendung)	Automatisch `System.out.println()`
Java-Thread	Code (Mandelbrot-Anwendung)	Automatisch `System.out.println()`
Java Thread Priorität	Code (Mandelbrot-Anwendung)	Automatisch `System.out.println()`
Affinität	CodeAnalyst (AMD)	Manuell Profile; Thread Profile
Affinität	ProcessExplorer (Sysinternals)	Manuell Performance Graph
CPU-Zeit	ProcessExplorer (Sysinternals)	Manuell Process view; CPU-Time
Kernel Thread Priorität	ProcessExplorer (Sysinternals)	Manuell Thread view; Base Priority
Kernel-Thread	ProcessExplorer (Sysinternals)	Manuell Process view; Threads
Kontextwechsel	ProcessExplorer (Sysinternals)	Manuell Thread view; Context Switches
Process Priority Class	ProcessExplorer (Sysinternals)	Manuell Process view; Priority
Verfügbare CPU	ProcessExplorer (Sysinternals)	Manuell Performance Graph

9.2.2. Ausgabe Testtools

Ausgabe Mandelbrot-Anwendung

Von der Mandelbrot-Anwendung werden wie folgt Daten auf die Konsole geschrieben:

```
. . . .
Calculating...
X: 1 to 1016
Y: 177 to 353
Thread Thread-1 done (Priority 5)
Thread Thread-4 done (Priority 5)
Thread Thread-2 done (Priority 5)
Thread Thread-3 done (Priority 5)
Run time: 4930ms
Number of pixels rendered: 718312
```

Listing 1 Ausgabe der Anwendung

Ausgabe Code Analyst

Ursprünglich sollte der „Intel Thread Profiler 3.0 for Windows“ ([2]) dazu benutzt werden, das Schedulingverhalten im Mehrprozessor-Umfeld zu analysieren. Da dieses viel versprechende Tool aber keine Unterstützung für die AMD Opteron-Architektur bietet, haben wir ein anderes Tool verwendet.

Mit dem „CodeAnalyst for Windows“ von AMD ([CODEANALYST]) steht mit der Version 2.69 (beta) eine kostenlose Alternative zu Verfügung.

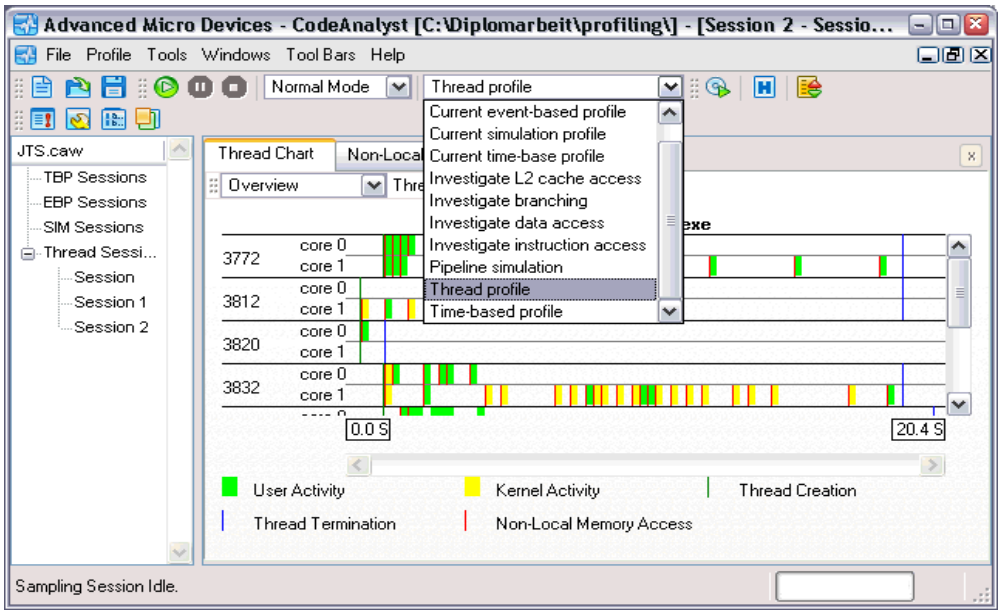

Abbildung 3 CodeAnalyst for Windows (AMD)

Ausgabe ProcessExplorer

Mit dem ProcessExplorer von Sysinternals steht wie erwartet ein sehr funktionales Tool zu Verfügung, mit dem „all-in-one“ eine Vielzahl von Performance Indikatoren gemessen werden können.

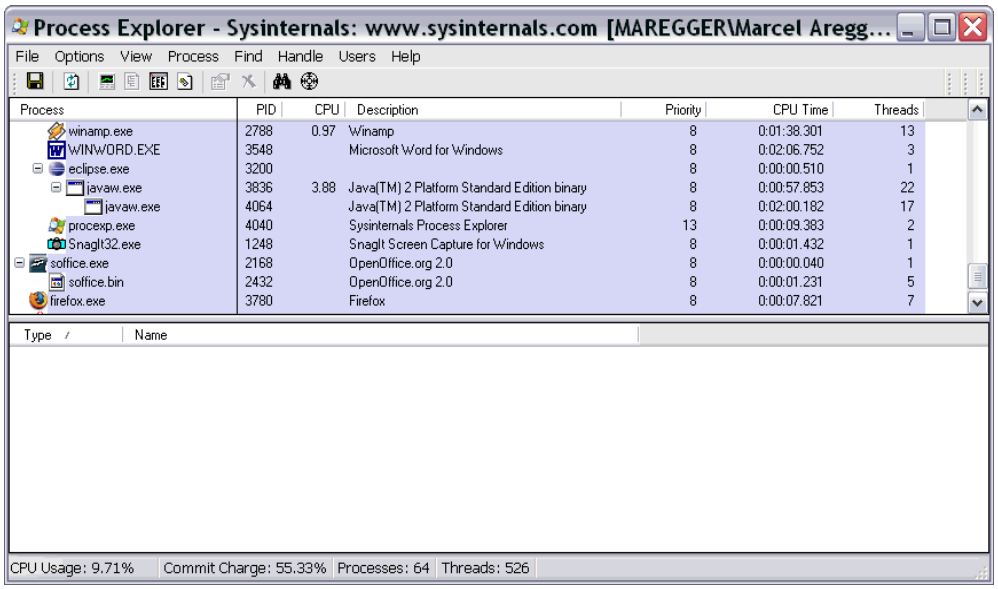

Abbildung 4 ProcessExplorer (Sysinternals)

10. Ergebnisse der Testcases

10.1. Testcase 1

Testcase 1	Betrachtungsbereich: Hardware
Zielsetzung	Feststellung Grad der Skalierung zwischen Single- und Multi-Prozessor-Architektur

10.1.1. Berechnungszeit

Tabelle 11 Grad der Skalierung – 1CPU/2CPU (Berechnungszeit)

	1 Thread	2 Thread
Berechnungszeit - 1CPU	69.41	69.77
Berechnungszeit - 2 CPU	68.10	35.72
Kontextwechsel - 1CPU	3555	3915
Kontextwechsel - 2CPU	13299	11014
Faktor Skalierung (zw. 1CPU/2CPU)	1.02	1.95

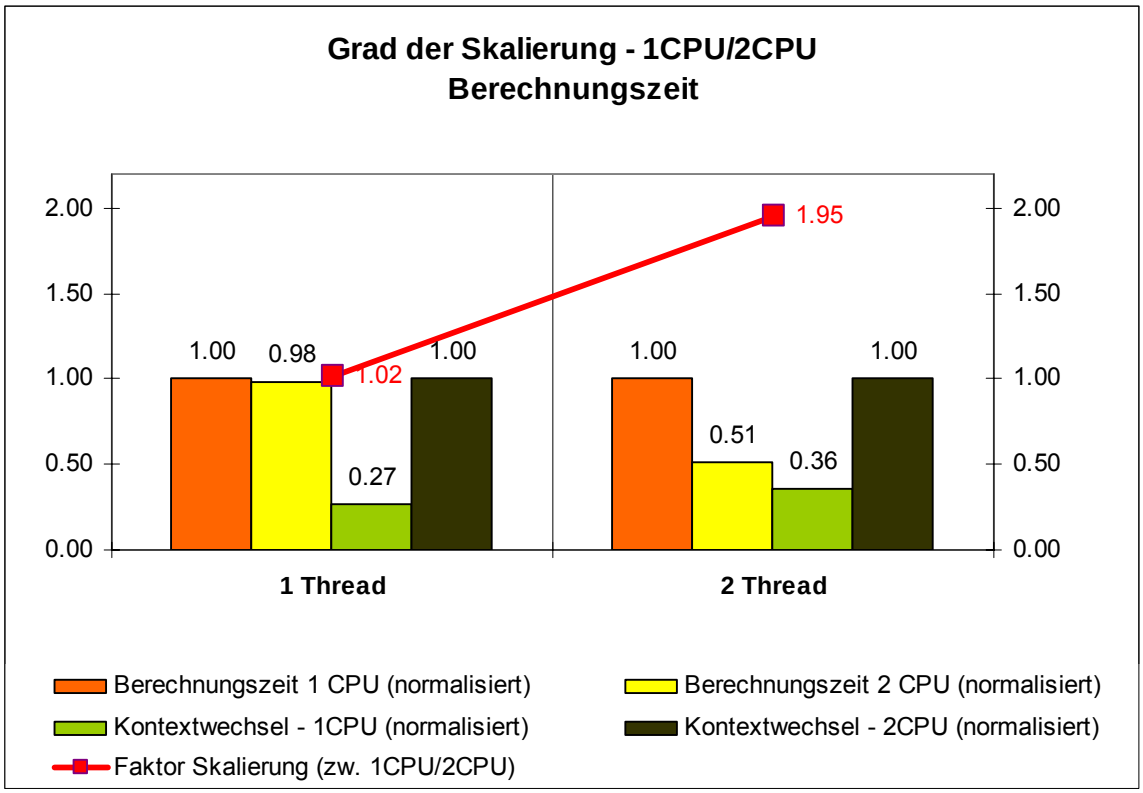

Abbildung 5 Grad der Skalierung – 1CPU/2CPU (Berechnungszeit)

10.1.2. CPU-Zeit

Tabelle 12 Grad der Skalierung 1CPU/2CPU (CPU-Zeit)

	1 Thread	2 Threads
CPU-Zeit - 1 CPU	1:09	1:09
CPU-Zeit - 2 CPU	1:12	1:11
Kontextwechsel - 1 CPU	3555	3915
Kontextwechsel - 2 CPU	13299	11014
Faktor Skalierung (zw. 1CPU/2CPU)	0.96	0.97

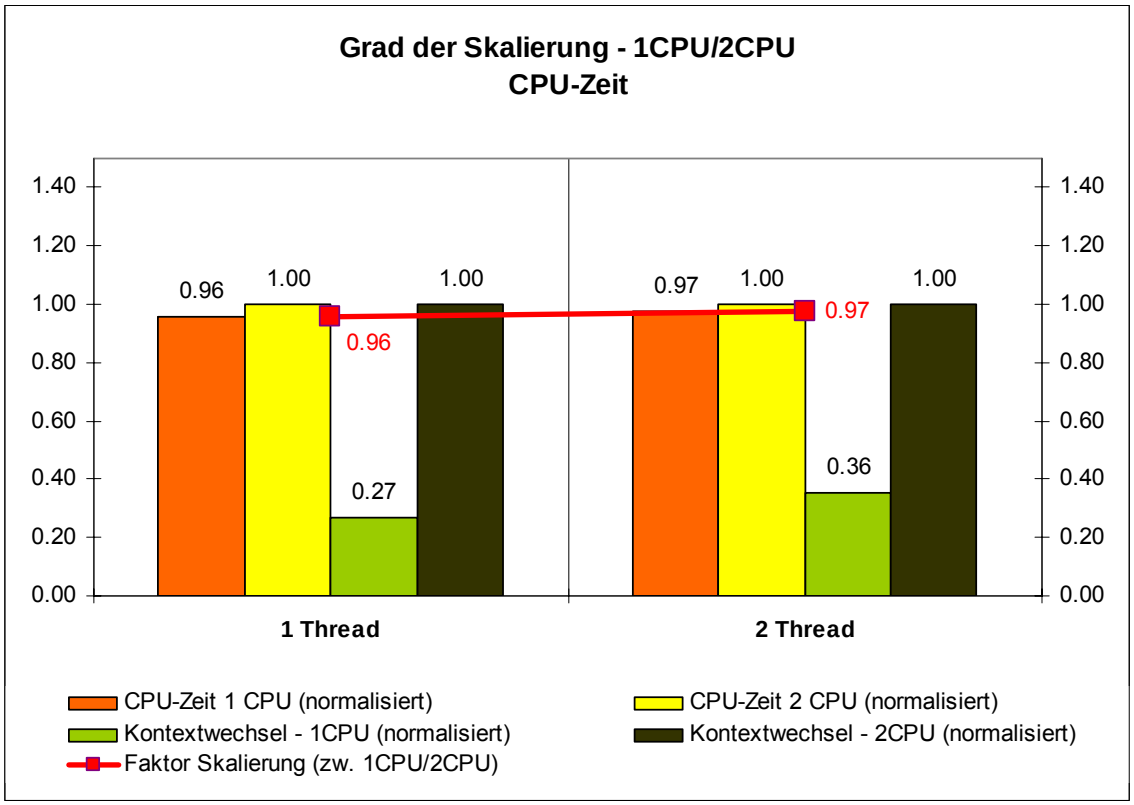

Abbildung 6 Grad der Skalierung 1CPU/2CPU (CPU-Zeit)

10.1.3. Ergebnisse

Für den Grad der Skalierung zwischen Single- und Multi-Prozessor-Architektur (Testcase 1) können aus den Messresultaten und dem zugehörigen Graph (Abbildung 6) folgende Eigenschaften abgeleitet werden:

- Berechnungszeit mit 1 Thread ist gleich bleibend
- Berechnungszeit mit 2 Thread auf 2 CPU wird halbiert (Skalierungsfaktor 1.95)
- Sprunghafter Anstieg für die Anzahl Kontextwechsel auf zwei CPU (1/2 Threads)

10.2. Testcase 2

Testcase 2	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Nachweis Abbildung Java-Thread auf Win32-Thread

10.2.1. Thread Mapping

Tabelle 13 Thread Mapping

# Java Threads	# Kernel Threads	# Deamon Threads	PID Process
0	13	13	2572
1	14	13	2572
3	16	13	2572
16	29	13	2572
32	45	13	2572
(Grenzwert) 7146	7159	13	2572

10.2.2. Ergebnisse

Mit der flexiblen Mandelbrot-Anwendung und dem Processexplorer konnte die Erzeugung von Worker- und Deamon-Threads im Kern einfach verfolgt werden. Für die Beziehung zwischen Java- und Kernel-Threads unter Windows XP gelten folgende Eigenschaften:

- 1:1 Abbildung zwischen Java- und Kernel-Threads
- Mit variabler Anzahl Worker-Threads werden 13 Deamon Threads erzeugt
- Für die bestehende Plattform resultiert ein Grenzwert von 7146 Threads
- Zur Laufzeit erzeugte Threads werden ebenfalls auf Kernel-Threads abgebildet

10.3. Testcase 3

Testcase 3	Betrachtungsbereich: Betriebssystem
Zielsetzung	Nachweis Abbildung Java-Thread-Priorität auf Win32-Thread-Priorität mit unveränderter Default Process Priority-Class

10.3.1. Priority-Mapping

Tabelle 14 Priority Mapping Java-/Win32 Thread (klein)

Java Thread Priorität	1	2	3	4	5	6	7	8	9	10
Win32-Thread-Priorität	6	6	7	7	8	8	9	9	10	10
	Process Priority Class = 8 (Normal)									

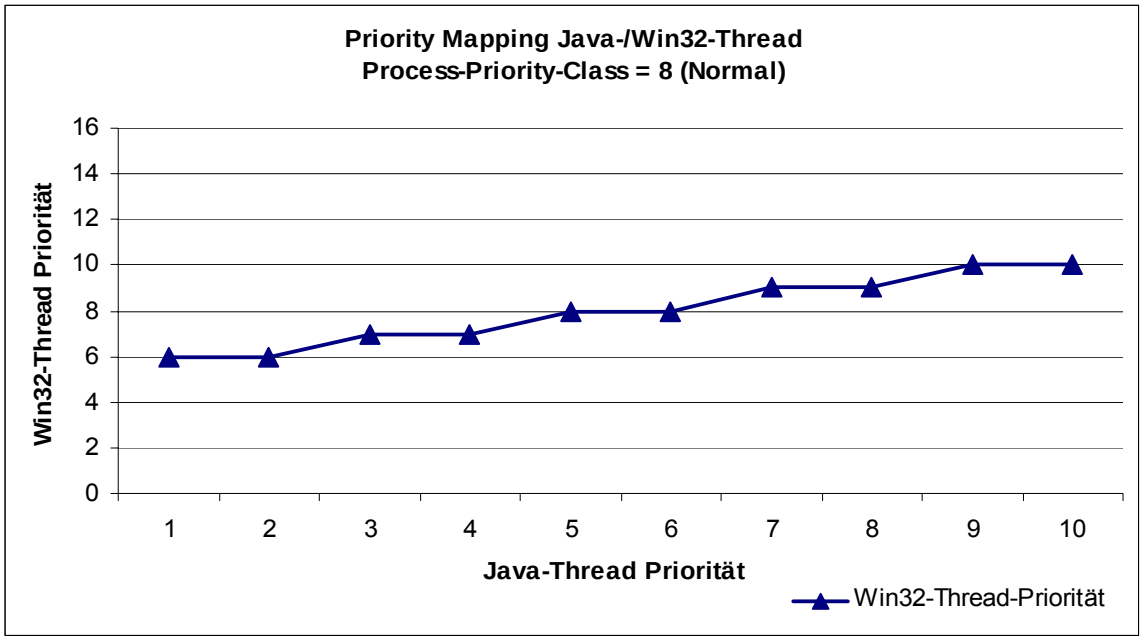

Abbildung 7 Priority Mapping Java-/Win32 Thread (klein)

10.3.2. Ergebnisse

Die Java-Thread-Priorität wird von der JVM auf eine entsprechende Kernel-Priorität gemappt. Für dieses Mapping gelten folgende Regeln:

- Java Prioritäten werden linear auf Basisprioritäten abgebildet
- Mehrere Java Prioritäten werden auf die selbe Basispriorität abgebildet
- Faktisch stehen nur 5 Java Prioritäten zu Verfügung

10.4. Testcase 4

Testcase 4	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Analyse Systemverhalten bei Änderung Win32-Thread-Priorität mit variabler Default Process Priority-Class

10.4.1. Priority-Mapping

Tabelle 15 Priority Mapping Java-/Win32 Thread (gross)

Process Priority Class	Java Thread Priorität				
	1	2	3	4	5
IDLE_PRIORITY_CLASS	2	2	3	3	4
BELOW_NORMAL_PRIORITY_CLASS	4	4	5	5	6
NORMAL_PRIORITY_CLASS	6	6	7	7	8
ABOVE_NORMAL_PRIORITY_CLASS	8	8	9	9	10
HIGH_PRIORITY_CLASS	11	11	12	12	13
REALTIME_PRIORITY_CLASS	22	22	23	23	24

Process Priority Class	Java Thread Priorität				
	6	7	8	9	10
IDLE_PRIORITY_CLASS	4	5	5	6	6
BELOW_NORMAL_PRIORITY_CLASS	6	7	7	8	8
NORMAL_PRIORITY_CLASS	8	9	9	10	10
ABOVE_NORMAL_PRIORITY_CLASS	10	11	11	12	12
HIGH_PRIORITY_CLASS	13	14	14	15	15
REALTIME_PRIORITY_CLASS	24	25	25	26	26

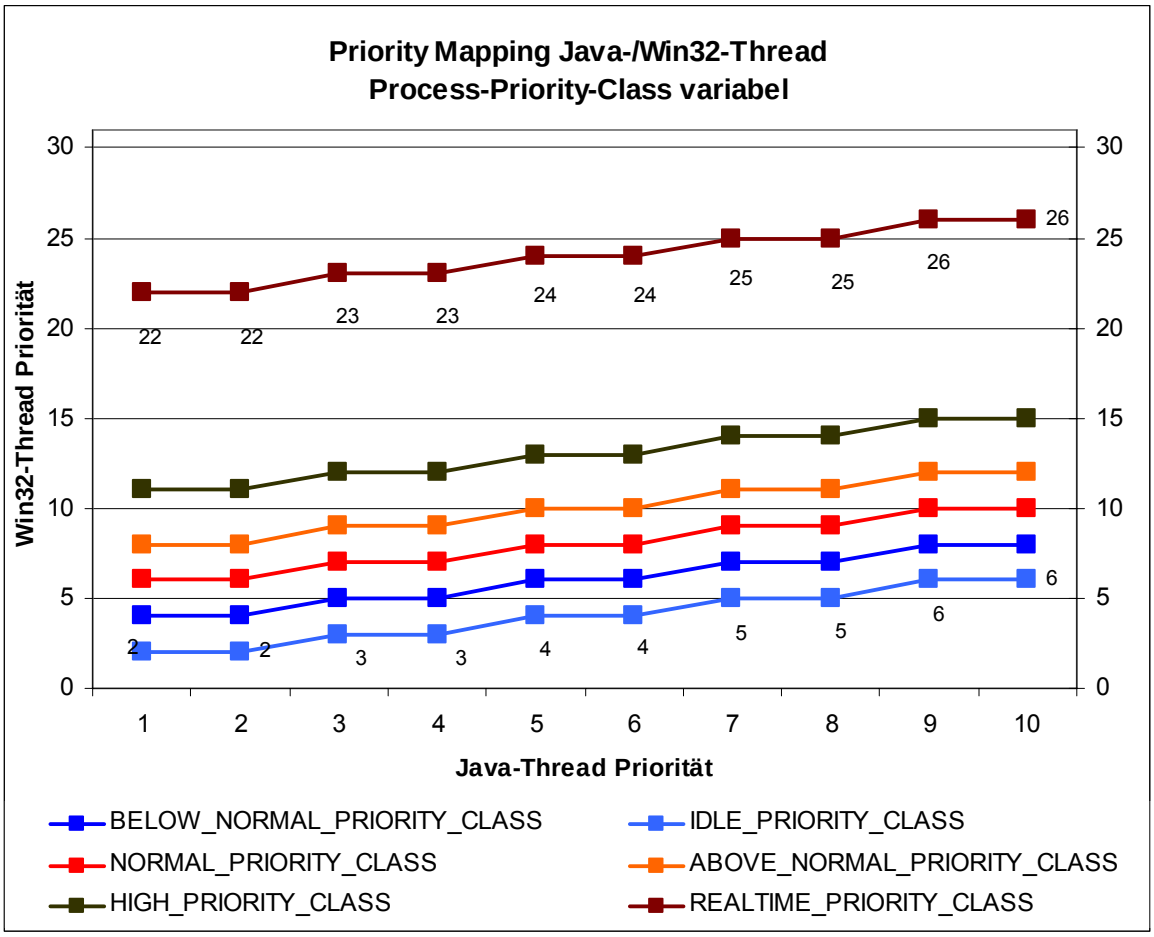

Abbildung 8 Priority Mapping Java-/Win32 Thread (gross)

10.4.2. Ergebnisse

Das Java-Priority-Mapping kann ausgedehnt werden, wenn zusätzlich die Kernel Process-Priority-Class berücksichtigt wird. Die Matrix und der zugehörige Graph (Abbildung 8) zeigen folgende charakteristische Eigenschaften:

- Java Prioritäten werden linear auf Basisprioritäten abgebildet
- Mehrere Java Prioritäten werden auf die selbe Basispriorität abgebildet
- Faktisch stehen nur 5 Java Prioritäten pro Process Priority Class zu Verfügung
- Überproportionaler Sprung für die HIGH_PRIORITY_CALSS (Δ=3)
- Überproportionaler Sprung für die REAL_TIME_PRIORITY_CLASS (Δ=11)
- Überlappungen Basisprioritäten bei unterschiedlichen Process Priority Classes
- Standard Basispriorität ist 8 (Standard Java Priorität = 5, NORMAL_PRIORITY_CLASS)
- (Bemerkung; nur Admin kann mit REAL_TIME_PRIORITY_CLASS Basis-Prioritäten über 15 erzeugen)

10.5. Testcase 5

Testcase 5	Betrachtungsbereich: Betriebssystem
Zielsetzung	Analyse Systemverhalten (Skalierung) bei Änderung der Process Priority Class und unterschiedlicher Last

10.5.1. Berechnungszeit

Tabelle 16 Skalierung mit variabler Process Priority Class (Berechnungszeit)

	Process Priority Class	
Win32-Thread-Prio=8	NORMAL_PRIORITY_CLASS	ABOVE_NORMAL_PRIORITY_CLASS
Laststufe null	35.7358	37.7074
Laststufe klein	42.0374	46.7240
Laststufe gross	47.0906	49.8650
Faktor Skalierung (Last klein/gross)	0.89	0.94

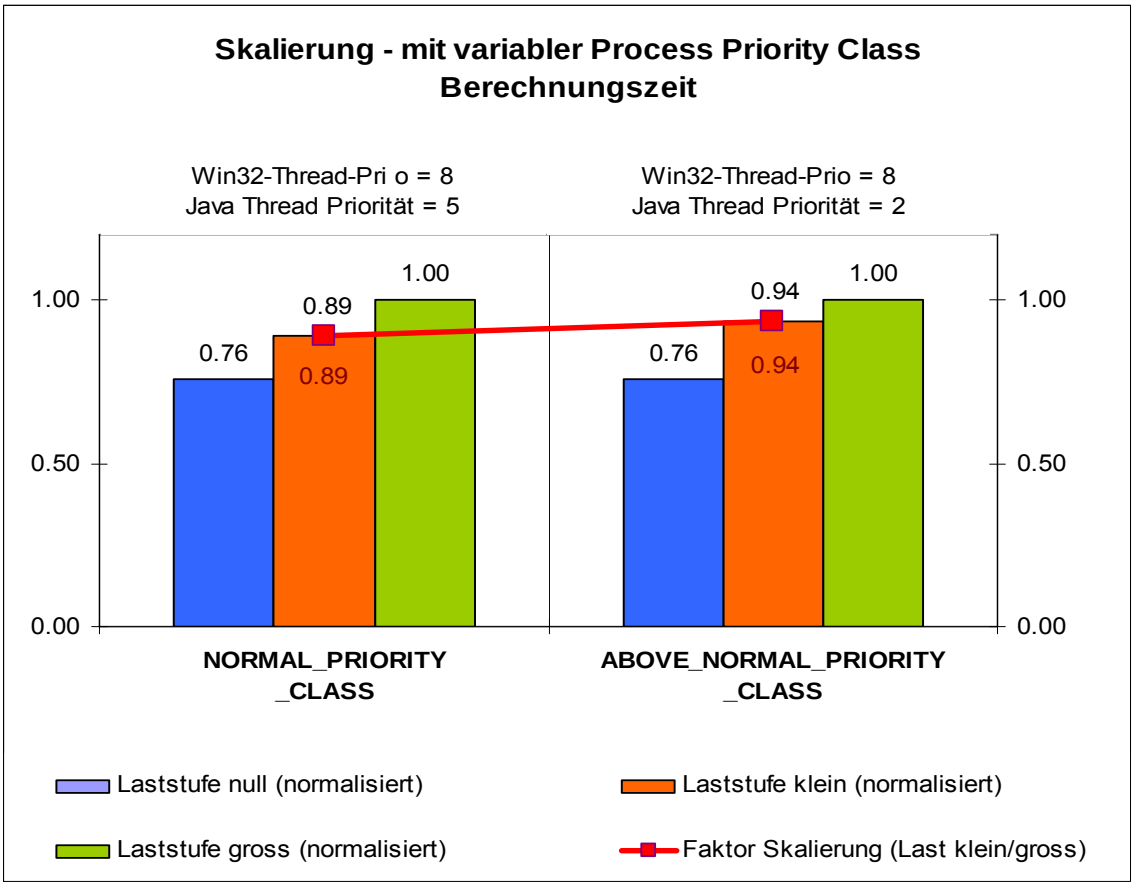

Abbildung 9 Skalierung mit variabler Process Priority Class (Berechnungszeit)

10.5.2. CPU-Zeit

Tabelle 17 Skalierung mit variabler Process Priority Class (CPU-Zeit)

	Process Priority Class	
Win32-Thread-Prio=8	NORMAL_PRIORITY_CLASS	ABOVE_NORMAL_PRIORITY_CLASS
Laststufe null	1:10	1:12
Laststufe klein	1:09	1:13
Laststufe gross	1:09	1:12
Faktor Skalierung (Last klein/gross)	1.00	1.01

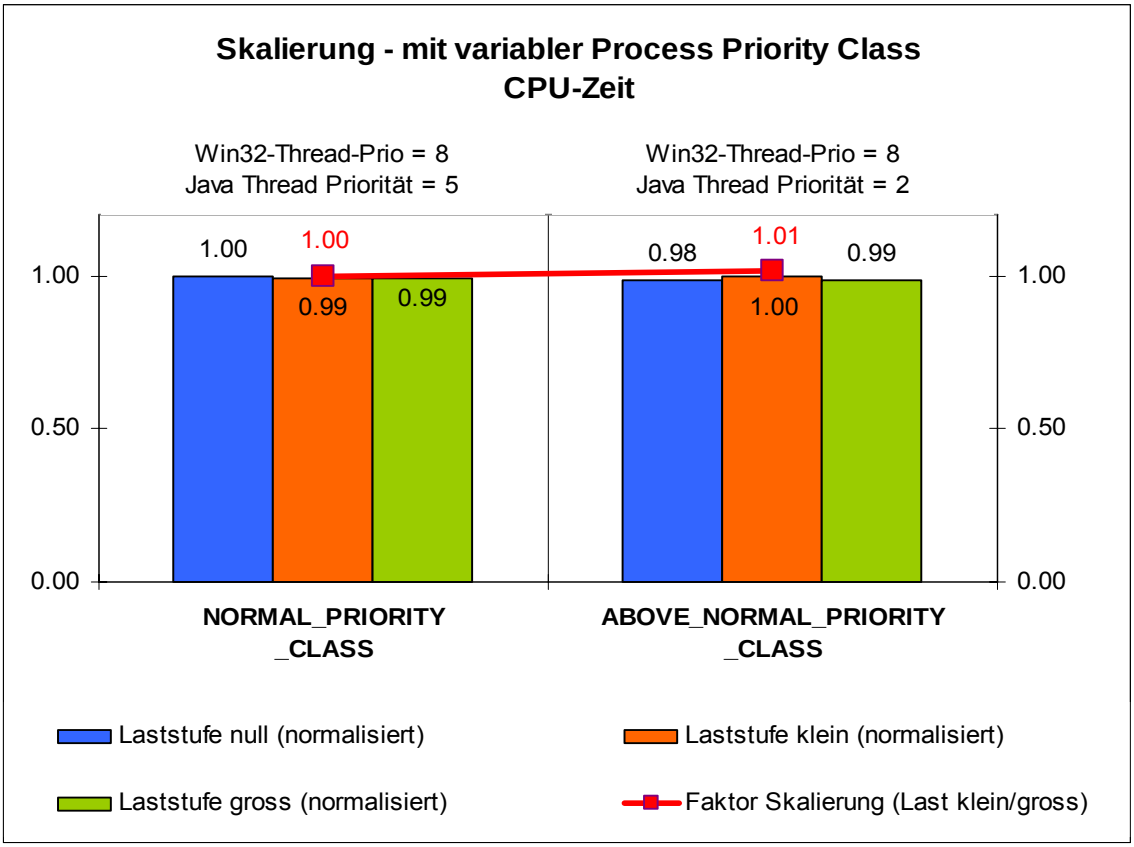

Abbildung 10 Skalierung mit variabler Process Priority Class (CPU-Zeit)

10.5.3. Ergebnisse

Der Einfluss einer variablen Process Priority Class auf das Schedulingverhalten des Kernels bei gleichbleibender Basispriorität kann wie folgt umschrieben werden:

- Für gleiche Basispriorität resultiert die gleiche Berechnungsdauer
- Die resultierende Berechnungsdauer ist unabhängig von der Process Priority Class
- Das Windows Scheduling wird nur durch die Basispriorität beeinflusst
- Die konsumierte CPU-Zeit bleibt konstant

10.6. Testcase 6

Testcase 6	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Analyse Systemverhalten bei Änderung Win32-Thread-Priorität und unterschiedlicher Last

10.6.1. Berechnungszeit (variable Win32-Priorität)

Tabelle 18 Skalierung mit variabler Win32-Priorität (Berechnungszeit)

	Win32-Thread-Priorität	
	Basispriorität 6	Basispriorität 10
Laststufe null	37.51	34.85
Laststufe klein	72.29	34.71
Laststufe gross	100.00	34.99
Faktor Skalierung (Last klein/gross)	0.72	0.99

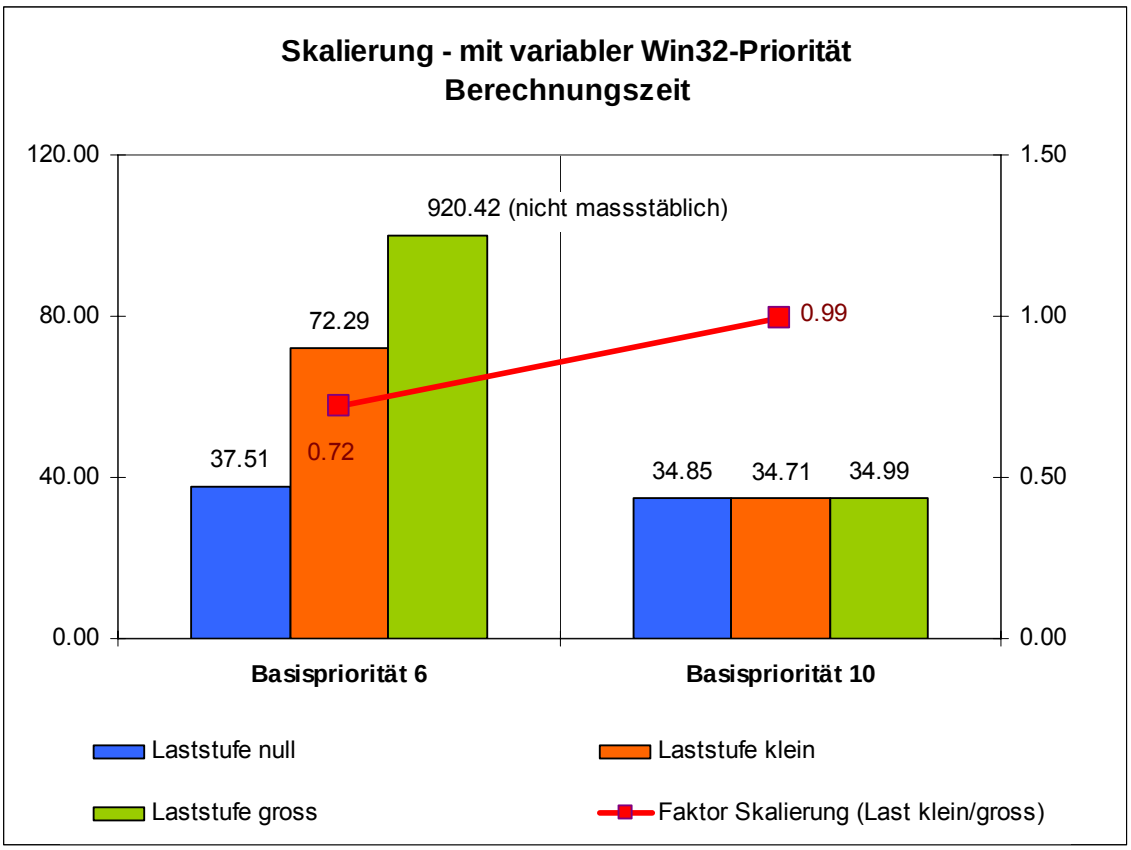

Abbildung 11 Skalierung mit variabler Win32-Priorität (Berechnungszeit)

10.6.2. Berechnungszeit (variable Laststufe)

Tabelle 19 Skalierung mit variabler Laststufe (Berechnungszeit)

Berechnungszeit (variable Basispriorität und Laststufe)

	Laststufe null	Laststufe klein	Laststufe gross
Basispriorität 6	37.51	72.29	100.00
Basispriorität 8	35.74	42.04	47.09
Basispriorität 10	34.85	34.71	34.99

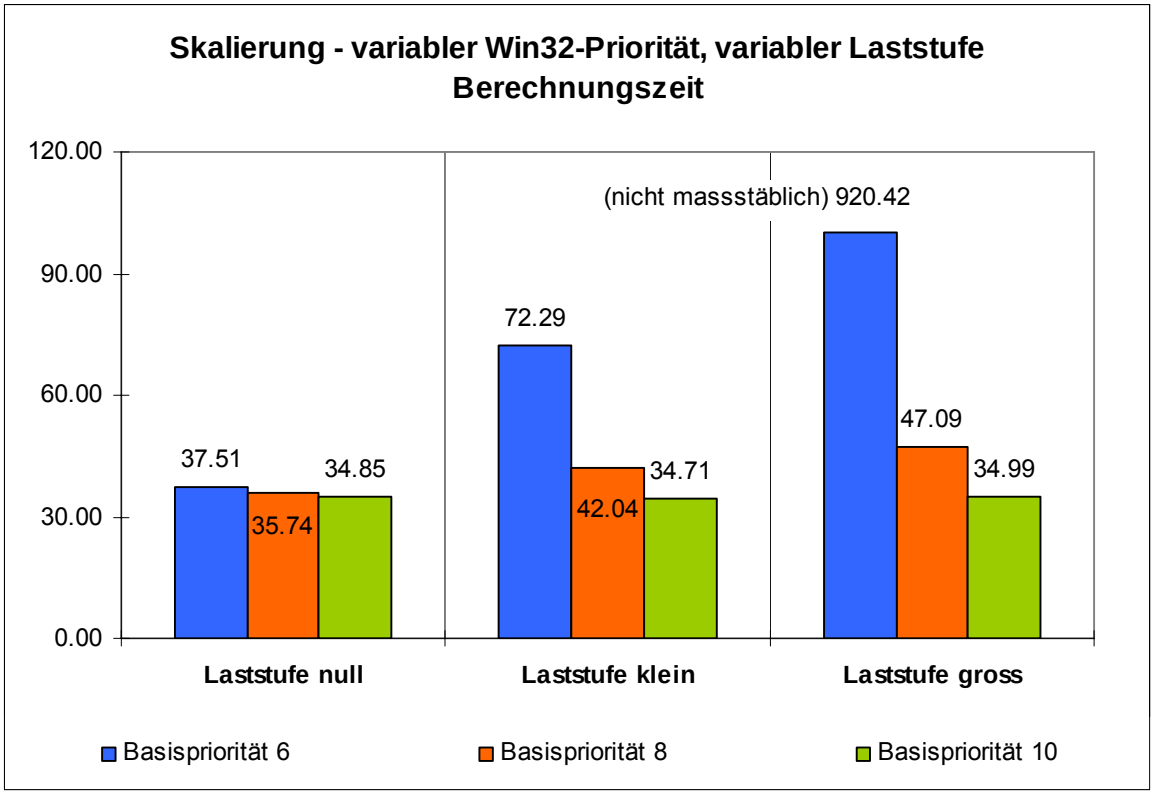

Abbildung 12 Skalierung mit variabler Laststufe (Berechnungszeit)

10.6.3. CPU-Zeit

Tabelle 20 Skalierung mit variabler Win32-Priorität (CPU-Zeit)

	Win32-Thread-Priorität	
	Basispriorität 6	Basispriorität 10
Laststufe null	1:12	1:08
Laststufe klein	1:14	1:08
Laststufe gross	1:09	1:08
Faktor Skalierung (Last klein/gross)	1.08	1.00

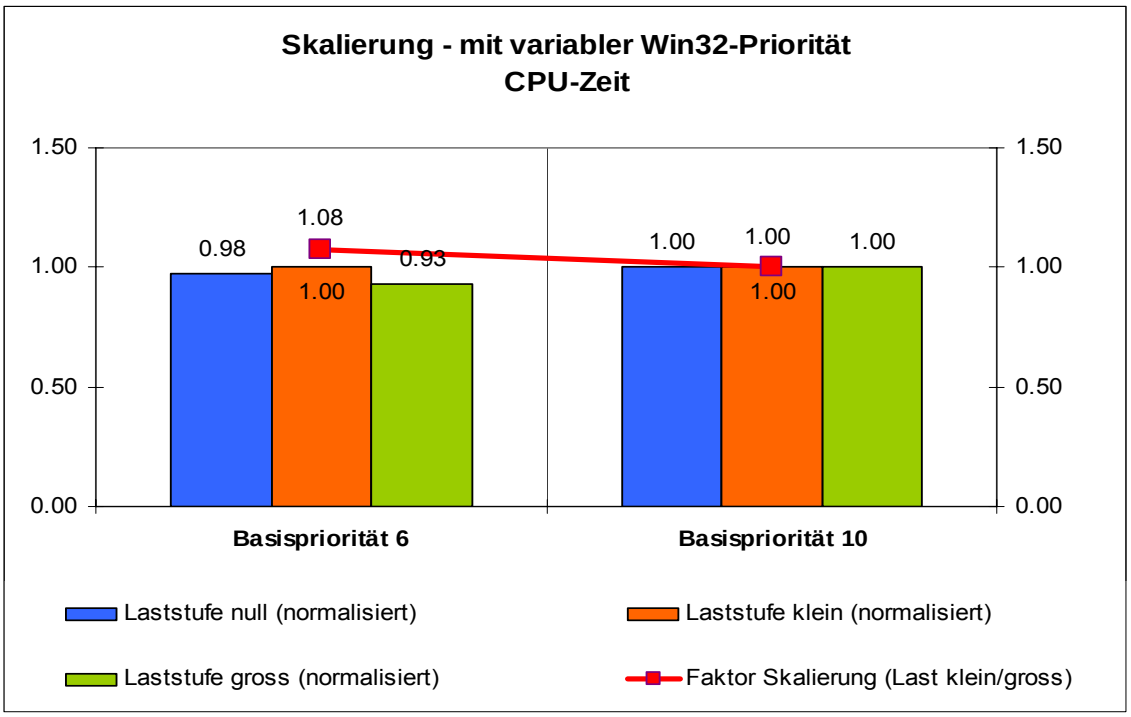

Abbildung 13 Skalierung mit variabler Win32-Priorität (CPU-Zeit)

10.6.4. Ergebnisse

Der Einfluss einer variablen Basispriorität (Win32-Thread-Priorität) auf das Schedulingverhalten des Kernels kann abhängig von 3 verschiedenen Laststufen wie folgt umschrieben werden:

- Für Laststufe „null“ resultiert die gleiche Berechnungszeit, die Basispriorität ist nicht relevant
- Für Laststufe „gering“ ergibt sich mit tieferer Priorität in Bezug auf die Last eine Verdoppelung der Berechnungszeit
- Für Laststufe „gross“ ergibt sich mit tieferer Priorität in Bezug auf die Last einen sprunghaften Anstieg der Berechnungszeit
- Für alle Laststufe ergeben sich mit höherer Priorität in Bezug auf die Last gleichbleibende Berechnungszeiten

10.7. Testcase 7

Testcase 7	**Betrachtungsbereich: Betriebssystem**
Zielsetzung	Analyse Systemverhalten bei Festlegung einer Prozess-Affinität

Tabelle 21 Java Anwendung ohne Affinität

Testcase	# Java Threads	Basispriorität	Affinität	Last	Basispriorität	Affinität
Rahmenbedingung	2	8	ohne	NEIN	keine	ohne
Berechnungszeit	35.5900					

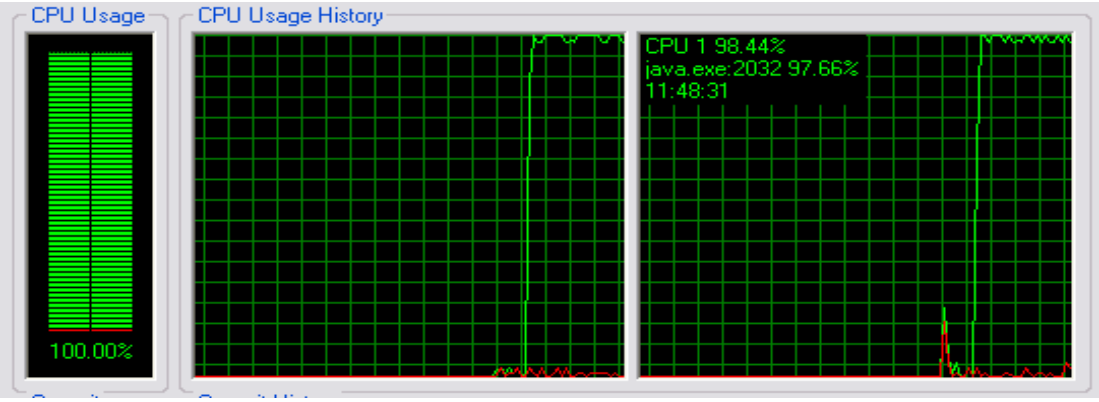

Abbildung 14 Java Anwendung ohne Affinität

Tabelle 22 Java Anwendung - Affinität auf CPU 1

Testcase	# Java Threads	Basispriorität	Affinität	Last	Basispriorität	Affinität
Rahmenbedingung	2	8	CPU 1	NEIN	keine	ohne
Berechnungszeit	69.8000					

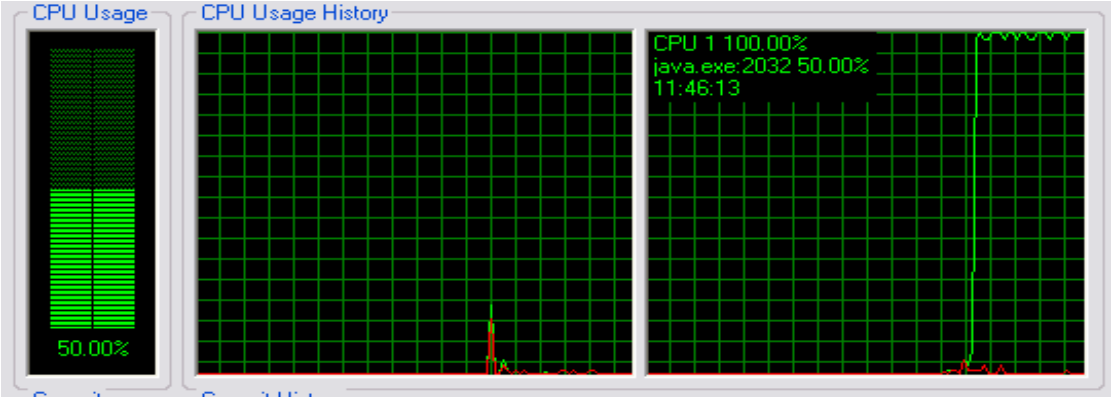

Abbildung 15 Java Anwendung - Affinität auf CPU 1

Tabelle 23 Calc ohne Affinität; Java Anwendung - Affinität auf CPU 1

Testcase	# Java Threads	Basispriorität	Affinität	Last	Basispriorität	Affinität
Rahmenbedingung	2	8	CPU 1	JA	8	ohne
Berechnungszeit	101.3900					

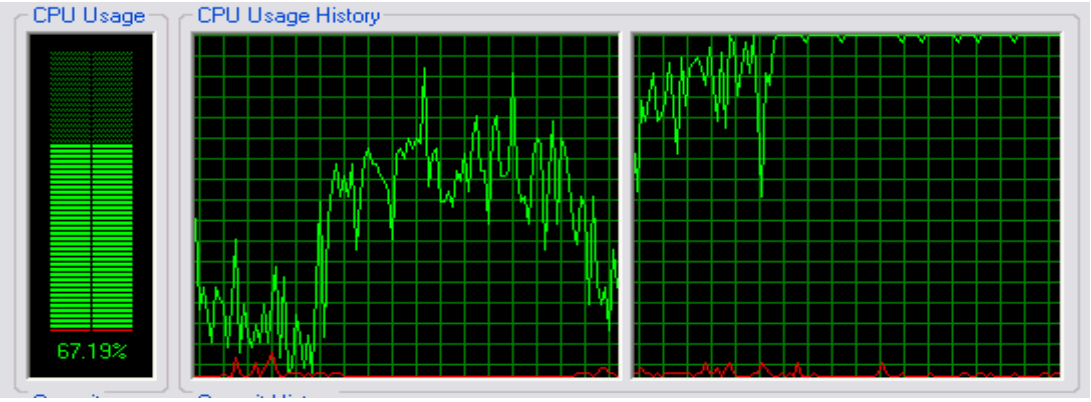

Abbildung 16 Calc ohne Affinität; Java Anwendung - Affinität auf CPU 1

Tabelle 24 Calc - Affinität CPU 0; Java Anwendung - Affinität auf CPU1

Testcase	# Java Threads	Basispriorität	Affinität	Last	Basispriorität	Affinität
Rahmenbedingung	2	8	CPU 1	JA	8	CPU 0
Berechnungszeit	70.2000					

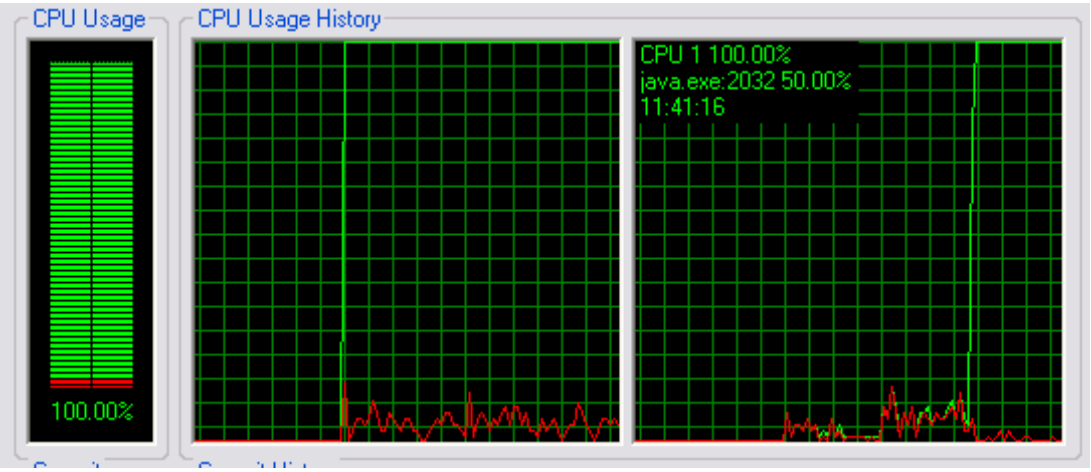

Abbildung 17 Calc - Affinität CPU 0; Java Anwendung - Affinität auf CPU1

10.7.1. Ergebnisse

Das Verhalten des Kernels bei expliziter Zuweisung eines Prozessors (Affinität) konnte mit Hilfe eines Systools analysiert werden und lässt sich wie folgt charakterisieren:

- Ohne Affinität und Last resultiert eine gleichförmige Verteilung auf alle verfügbaren CPUs

- Vererbung der Prozess Affinität auf die Threads dieses Prozesses
- Durch die Prozesses-Affinität werden andere Prozessoren für diesen Prozess „ausgeblendet“
- Ein Prozessor wird durch die Affinitäts(Maske) nicht exklusiv zugeteilt
- Die manuelle „Trennung“ von Anwendungen (Prozessen) ist über die Affinität möglich

10.8. Testcase 8

Testcase 8	Betrachtungsbereich: JVM
Zielsetzung	Analyse der Skalierung einer multithreaded Java-Applikation

10.8.1. Skalierung 1 CPU ohne Synchronisation

Tabelle 25 Skalierung 1 CPU ohne Synchronisation

1 CPU	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	69.6344	70.3050	70.6148	69.4788	69.7438	69.5392
CPU-Zeit	1:09	1:10	1:10	1:09	1:09	1:09
Faktor Skalierung	1.00	0.99	0.99	1.00	1.00	1.00

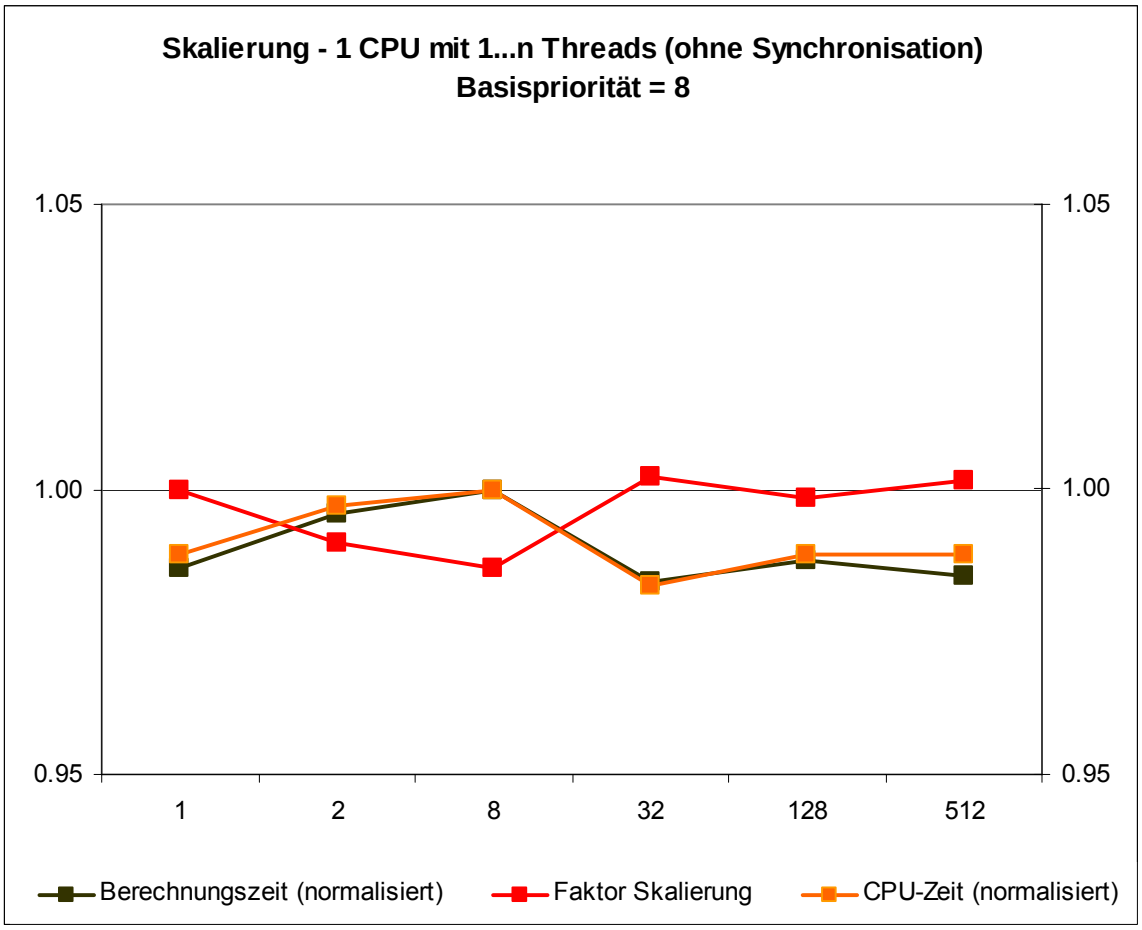

Abbildung 18 Skalierung 1 CPU ohne Synchronisation

10.8.2. Skalierung 2 CPU ohne Synchronisation

Tabelle 26 Skalierung 2 CPU ohne Synchronisation

2 CPU	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	68.2768	35.5212	35.5932	35.0078	35.3916	36.1504
CPU-Zeit	1:09	1:09	1:10	1:09	1:10	1:12
Faktor Skalierung	1.00	1.92	1.92	1.95	1.93	1.89

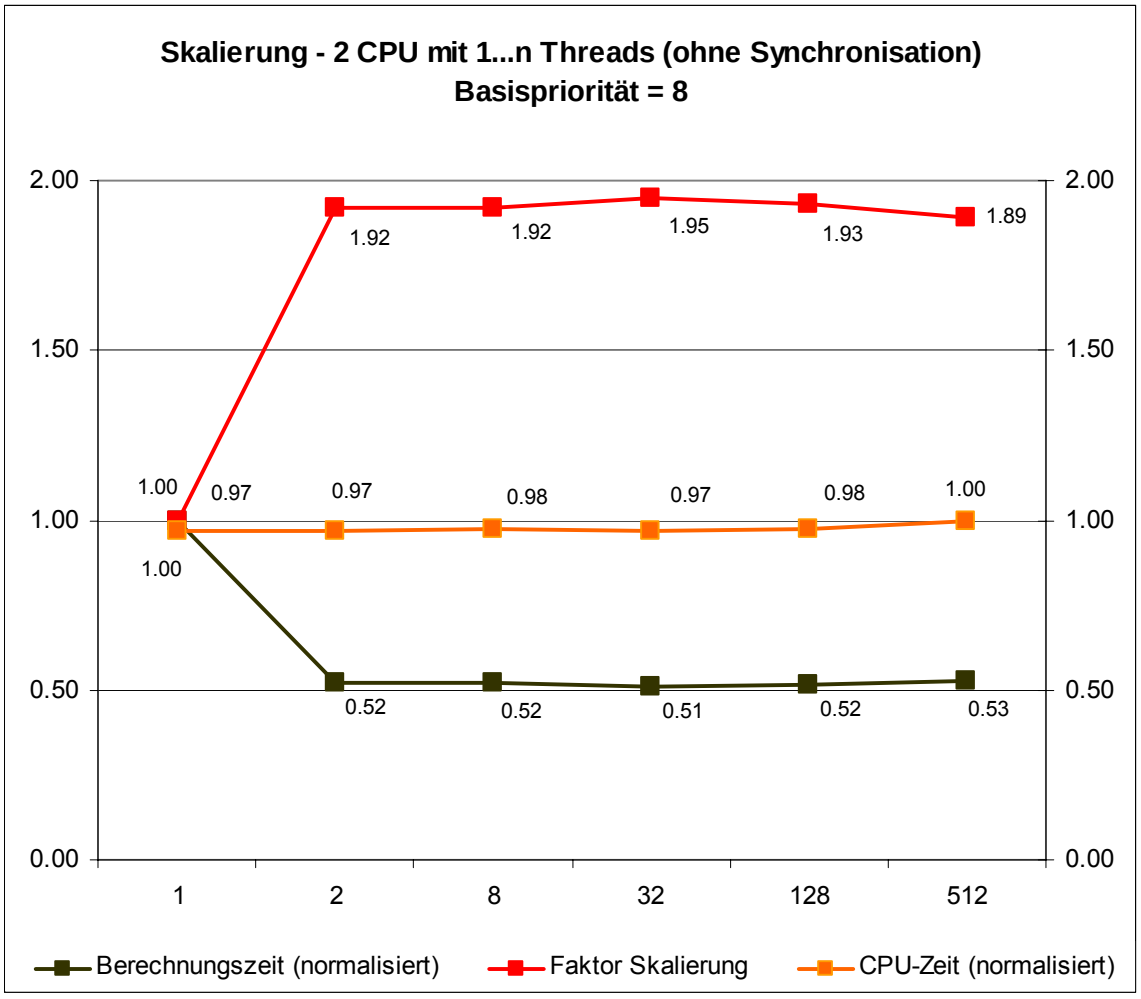

Abbildung 19 Skalierung 2 CPU ohne Synchronisation

10.8.3. Ergebnisse

Die Skalierung einer Anwendung mit variabler Anzahl Threads ohne Synchronisation zeigt zwischen der Single- und Multi-Prozessor-Architektur deutliche Unterschiede im Verlauf:

- Auf der 1 CPU-Architektur für 1 bis 512 Threads annähernd konstante Berechnungs- und CPU-Zeiten (Schwankung BZ: ~1.6%, Schwankung CPU-Zeit: ~1.4%)

- Auf der 2 CPU-Architektur ab 2 Threads durch Verteilung der Threads eine Halbierung der Berechnungszeit
- CPU-Zeit bleibt auch bei der 2 CPU-Architektur annähernd konstant (Schwankung CPU-Zeit: ~2.7%)
- Auf der 2 CPU-Architektur wird ab 2 Threads durch Verteilung der Threads ein Skalierungsfaktor von nahezu 2 erreicht
- Der zunehmende Verwaltungsaufwand für 1 bis 512 Threads führt bei der 1 CPU und 2 CPU-Architektur nicht zu einer nennenswerten Zunahme der CPU-Zeit

10.9. Testcase 9

Testcase 9	Betrachtungsbereich: JVM
Zielsetzung	Analyse Einfluss der Thread-Synchronisation auf die Skalierung

10.9.1. Skalierung 1 CPU - Methodensynchronisation

Tabelle 27 Skalierung 1 CPU mit Methodensynchronisation

1 CPU Methodensynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	69.3373	69.9843	114.2270	129.0133	127.3140	124.4367
CPU-Zeit	1:10	1:10	1:53	2:07	2:06	2:03
Faktor Skalierung	1.00	0.99	0.61	0.54	0.54	0.56

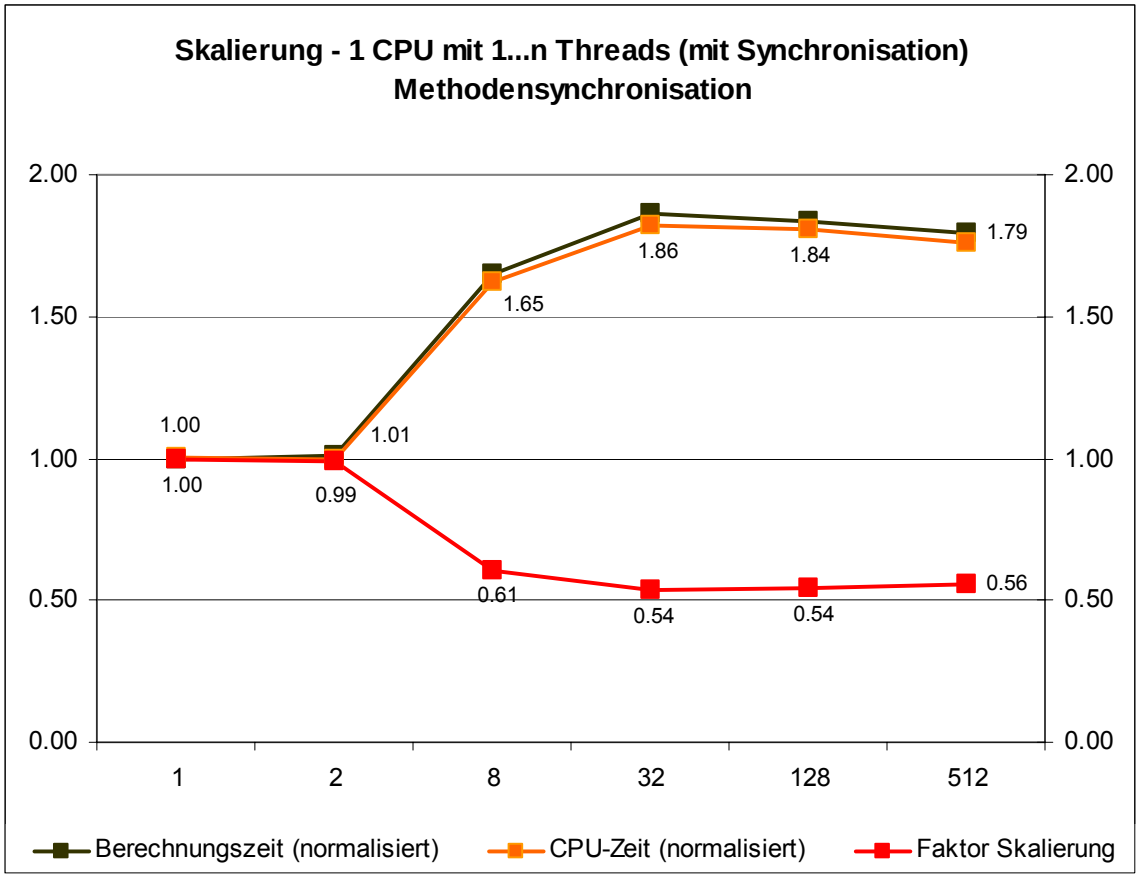

Abbildung 20 Skalierung 1 CPU mit Methodensynchronisation

10.9.2. Skalierung 1 CPU - Objektsynchronisation

Tabelle 28 Skalierung 1 CPU mit Objektsynchronisation

1 CPU Objektsynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	69.6983	69.6860	70.7043	69.8513	69.2120	70.2033
CPU-Zeit	1:10	1:10	1:11	1:10	1:09	1:10
Faktor Skalierung	1.00	1.00	0.99	1.00	1.01	0.99

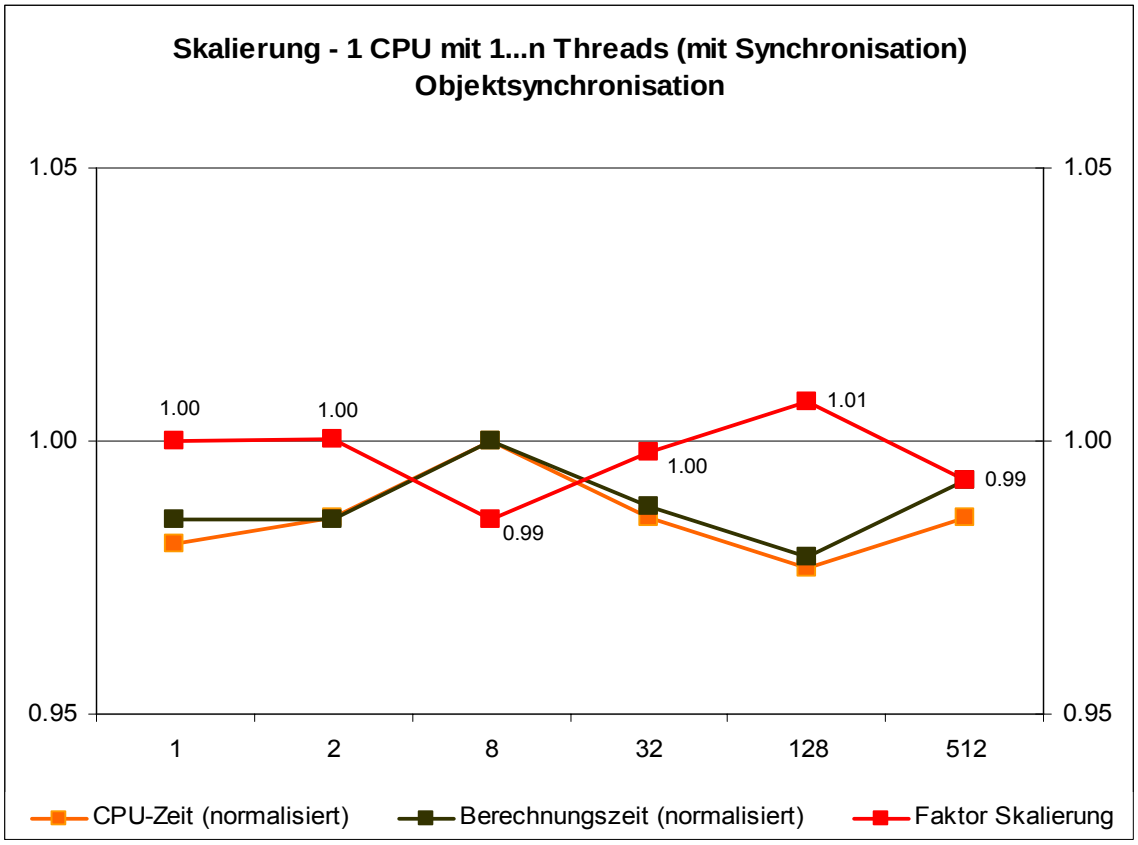

Abbildung 21 Skalierung 1 CPU mit Objektsynchronisation

10.9.3. Skalierung 1 CPU – CAS-Synchronisation

Tabelle 29 Skalierung 1 CPU mit CAS-Synchronisation

1 CPU CAS-Synchronisation	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	69.0243	69.5647	70.8340	69.7330	69.6180	69.6500
CPU-Zeit	1:09	1:10	1:11	1:10	1:10	1:10
Faktor Skalierung	1.00	0.99	0.97	0.99	0.99	0.99

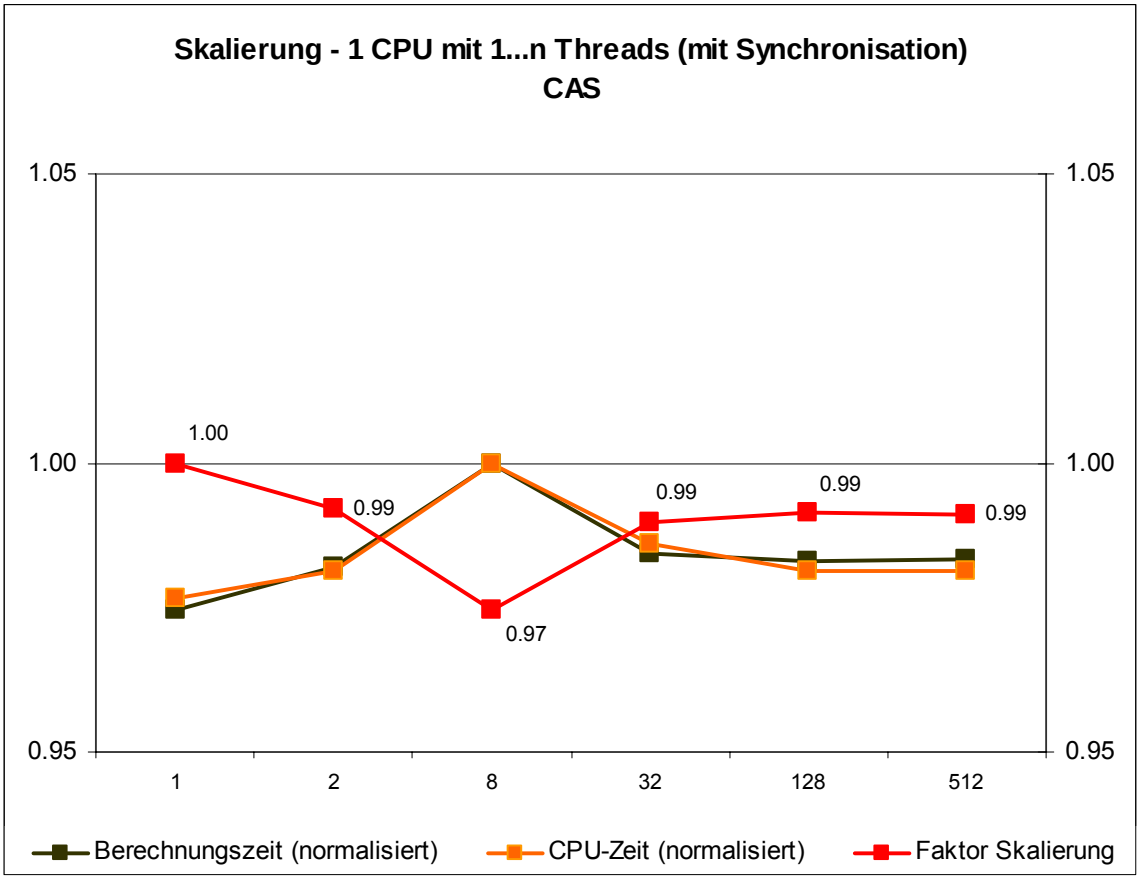

Abbildung 22 Skalierung 1 CPU mit CAS-Synchronisation

10.9.4. Ergebnisse

Die Skalierung einer Anwendung mit variabler Anzahl Threads zeigt auf einer 1 CPU-Architektur unter Berücksichtigung verschiedener Synchronisations-Methoden deutliche Unterschiede im Verlauf:

Methodensynchronisation

- Bis 2 Threads resultiert ein konstanter Verlauf von Berechnungs- und CPU-Zeit
- Verwendung > 2 Threads führt zu massiven Anstieg von Berechnungs- und CPU-Zeit
- Einbruch der Skalierung bei > 2 Threads um bis zu 50%

Objektsynchronisation

- Für Objektsynchronisation mit 1 bis 512 Threads annähernd konstante Berechnungs- und CPU-Zeit (Schwankung BZ: ~2.1%, Schwankung CPU-Zeit: ~2.8%)

CAS-Synchronisation

- Für CAS-Synchronisation mit 1 bis 512 Threads annähernd konstante Berechnungs- und CPU-Zeit (Schwankung BZ: ~2.6%, Schwankung CPU-Zeit: ~2.8%)

10.9.5. Skalierung 2 CPU - Methodensynchronisation

Tabelle 30 Skalierung 2 CPU mit Methodensynchronisation

2 CPU Methodensynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	69.3133	35.2490	38.5930	40.5993	42.1927	42.3203
CPU-Zeit	1:10	1:10	1:16	1:20	1:23	1:23
Faktor Skalierung	1.00	1.97	1.80	1.71	1.64	1.64

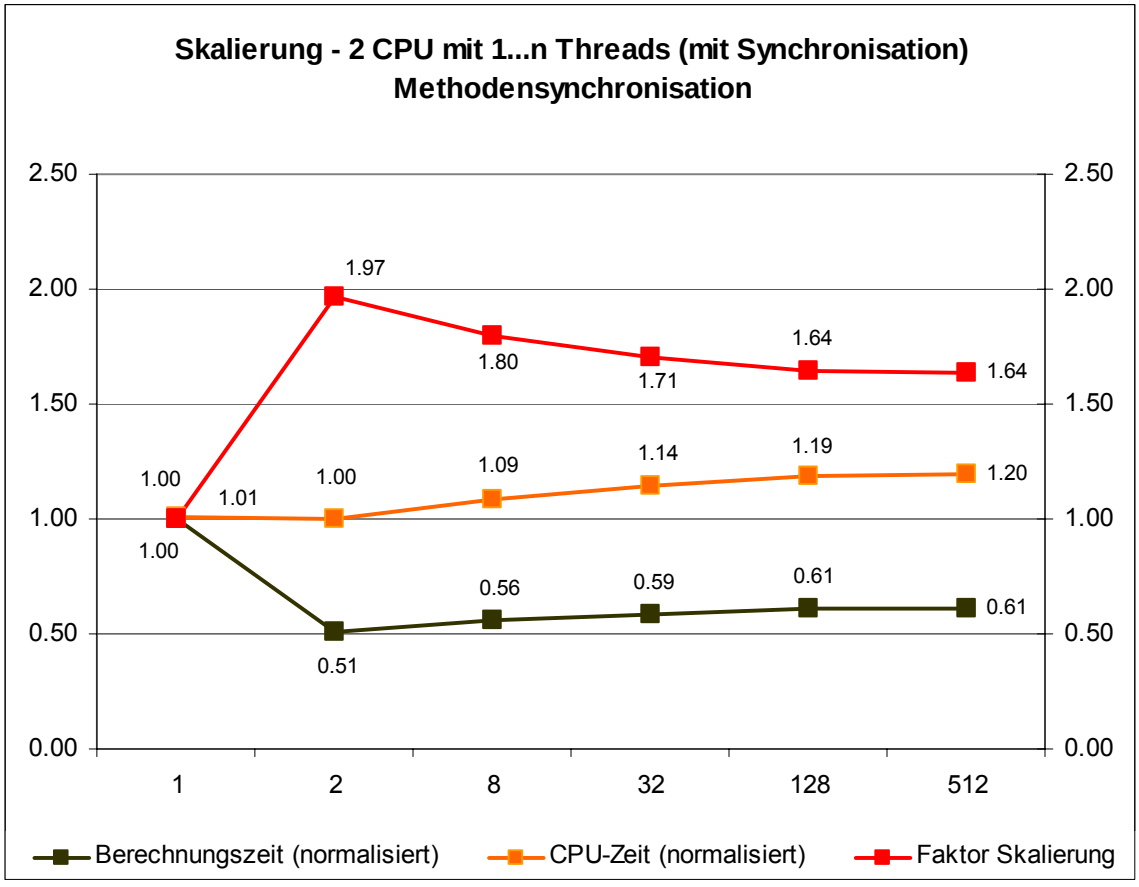

Abbildung 23 Skalierung 2 CPU mit Methodensynchronisation

10.9.6. Skalierung 2 CPU - Objektsynchronisation

Tabelle 31 Skalierung 2 CPU mit Objektsynchronisation

2 CPU Objektsynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	68.8343	35.8957	39.1343	40.4560	41.3240	42.1283
CPU-Zeit	1:13	1:11	1:17	1:20	1:21	1:23
Faktor Skalierung	1.00	1.92	1.76	1.70	1.67	1.63

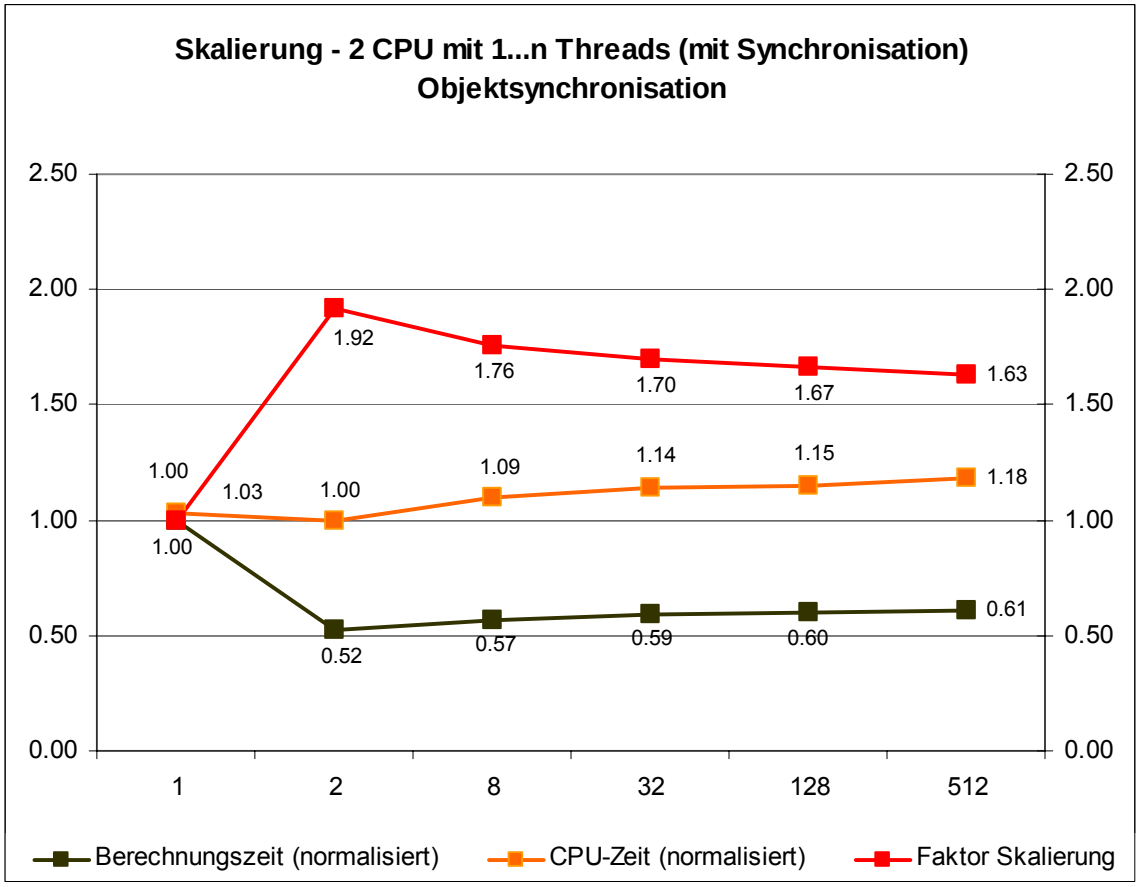

Abbildung 24 Skalierung 2 CPU mit Objektsynchronisation

10.9.7. Skalierung 2 CPU - CAS-Synchronisation

Tabelle 32 Skalierung 2 CPU mit CAS-Synchronisation

2 CPU CAS-Synchronisation	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	68.6270	35.9630	35.8297	35.1910	35.5630	37.0027
CPU-Zeit	1:10	1:10	1:11	1:10	1:11	1:14
Faktor Skalierung	1.00	1.91	1.92	1.95	1.93	1.85

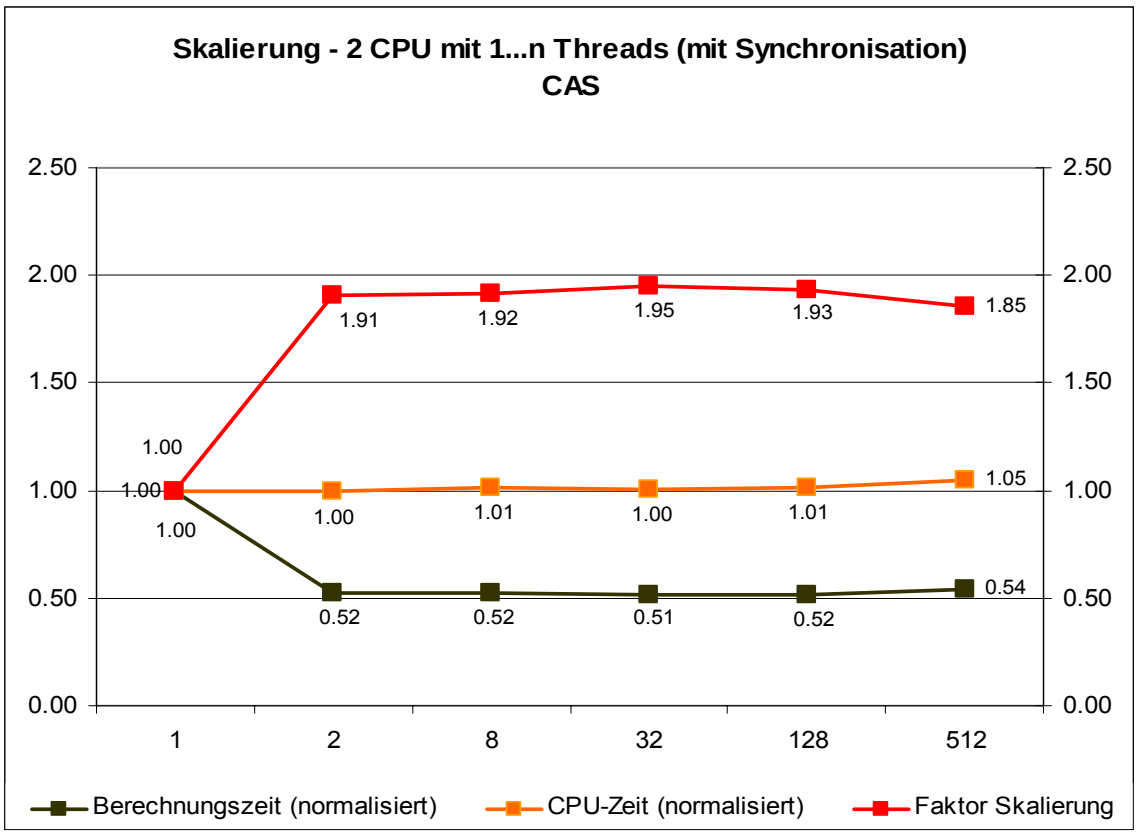

Abbildung 25 Skalierung 2 CPU mit CAS-Synchronisation

10.9.8. Ergebnisse

Die Skalierung einer Anwendung mit variabler Anzahl Threads zeigt auf einer 2 CPU-Architektur unter Berücksichtigung verschiedener Synchronisations-Methoden Ähnlichkeiten im Verlauf:

Methodensynchronisation

- Bis 2 Threads resultiert eine lineare Skalierung (Halbierung der Berechnungs-Zeit)
- Zwischen 3 bis 512 Threads ergibt sich eine 20%-Zunahme der BERECHNUNGS-Zeit
- Zwischen 3 bis 512 Threads ergibt sich eine 20% Zunahme der CPU-Zeit
- Maximaler Skalierungs-Faktor bei 2 Threads beträgt annähernd 2, sinkt bis 512 Threads auf ca.1.6

Objektsynchronisation

- Für Objektsynchronisation ähnlicher Verlauf im Vergleich zur Methoden Synchronisation
- Zwischen 3 bis 512 Threads ergibt sich eine 17%-Zunahme der Berechnungs-Zeit
- Zwischen 3 bis 512 Threads ergibt sich eine 18% Zunahme der CPU-Zeit
- Maximaler Skalierungs-Faktor bei 2 Threads beträgt annähernd 2, sinkt bis 512 Threads auf ca.1.6

CAS-Synchronisation

- Für CAS-Synchronisation flacherer Verlauf im Vergleich zur Methoden- und Objekt-Synchronisation
- Zwischen 3 bis 512 Threads ergibt sich eine 4%-Zunahme der Berechnungs-Zeit
- Zwischen 3 bis 512 Threads ergibt sich eine 5% Zunahme der CPU-Zeit
- Maximaler Skalierungs-Faktor bei 2 Threads beträgt 1.9, sinkt bis 512 Threads nur unwesentlich

Direkter Vergleich 1 CPU und 2CPU mit Methodensynchronisation

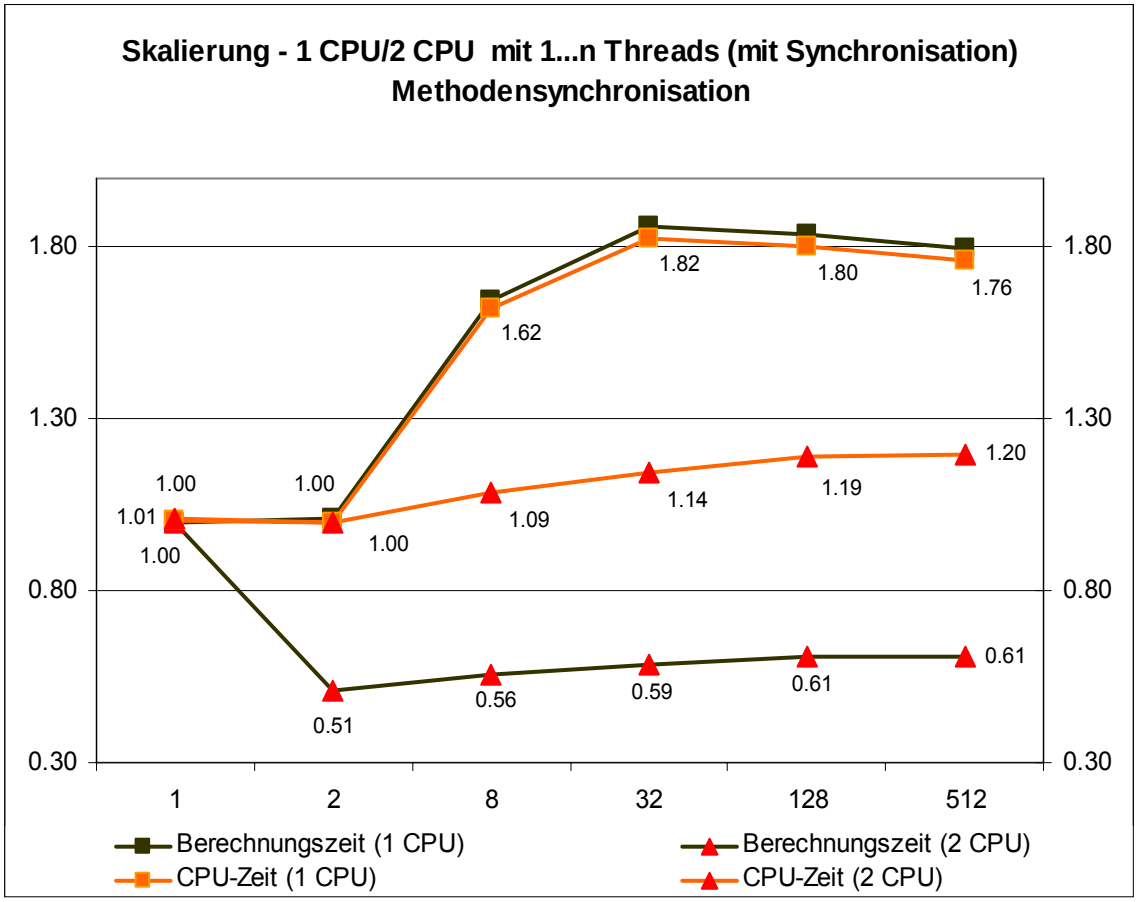

Abbildung 26 Skalierung 1 CPU/2 CPU mit Methodensynchronisation

Ergebnisse direkter Vergleich

- Ein 2 CPU-System wird durch grobkörnige Synchronisation deutlich weniger ausgebremst als ein 1 CPU-System
- Jede Synchronisation benötigt Rechenzeit, die im direkten Zusammenhang mit der Anzahl aktiver Threads und der resultierenden „lock-contention“ steht

10.10. Testcase 10

Testcase 10	Betrachtungsbereich: JVM
Zielsetzung	Parallelisierung durch JOMP

10.10.1. Skalierung 2 CPU - JOMP-Threads

Tabelle 33 Skalierung JOMP-Threads

2 CPU ohne Synchronisation	Anzahl JOMP-Threads					
Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	68.2590	36.0777	35.4657	35.2003	35.3213	35.7390
CPU-Zeit	1:10	1:14	1:11	1:13	1:15	1:15
Faktor Skalierung	1.00	1.89	1.92	1.94	1.93	1.91

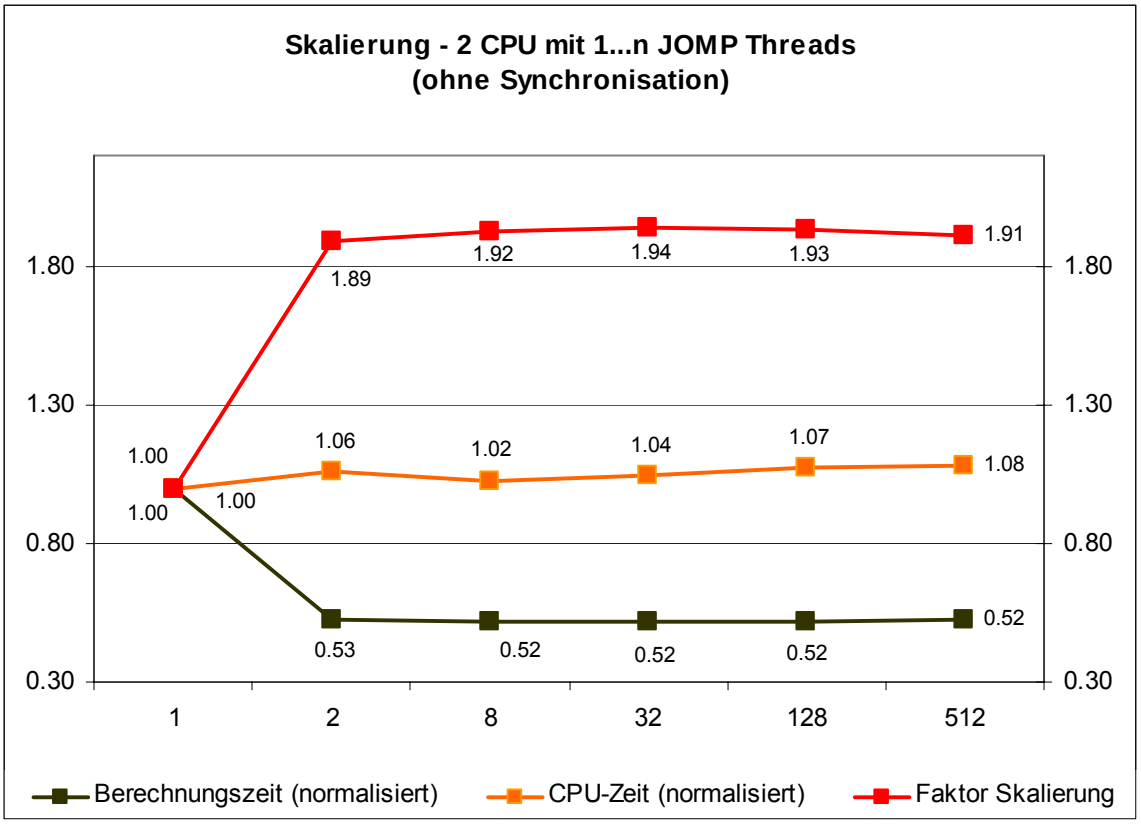

Abbildung 27 Skalierung JOMP-Threads

10.10.2. Ergebnisse

Die Skalierung einer JOMP-Anwendung mit variabler Anzahl Threads zeigt auf einer 2 CPU-Architektur unter Berücksichtigung der Methoden-Synchronisation folgende charakteristische Merkmale:

- Ab 2 Threads wird auf der 2 CPU-Architektur durch Verteilung der Threads eine Halbierung der Berechnungszeit erreicht
- CPU-Zeit bleibt auch bei JOMP-Threads annähernd konstant (Schwankung CPU-Zeit: ~1%)
- (Hinweis; Grafik zeigt leichten Anstieg der CPU-Zeit da Threads nach der Berechnung im Busy-Waiting verbleiben und somit CPU-Zeit verbrauchen)
- Skalierungsfaktor ist vergleichbar mit einer Java-Thread-Implementierung ohne Synchronisation (10.8.2 Skalierung 2 CPU ohne Synchronisation)

11. Glossar

Tabelle 34 Glossar

Begriff	Beschreibung
Affinität	Bezeichnet die Zuordnung eines Prozesses/Threads zu physikalischen Recheneinheiten. Durch die Definition einer Affinitätsmaske kann gesteuert werden auf welchen Recheneinheiten die Anwendung ausgeführt werden kann.
API	API (Application Programming Interface) defniert eine Schnittstelle zwischen verschiedenen Software Systemen. Eine API definiert typischerweise eine Reihe von Methoden, Parametern, Datentypen und Datenfeldern.
CAS	Compare and Swap bzw. Compare and Set bezeichnet eine atomare (meist hardwareunterstützte) Operation in der ein gespeicherter Wert mit dem vermuteten Wert verglichen wird. Stimmt dieser überein, so wird ein neuer Wert gesetzt. Ansonsten wird nichts getan. CAS Funktionen erlauben Lock-freie Algorithmen.
CPU	Abkürzung für Central Processing Unit. Wird synonym für die deutsche Bezeichnung Hauptrpozessor bzw. Prozessor verwendet.
DEP	Data Execution Prevention; Eine Technologie, die es erlaubt Speicherbereiche als nicht ausführbar zu markieren. Damit verlieren Overflow-Basierende Sicherheitslücken (die grosse Masse) an Bedeutung. AMD bezeichnet die Hardware-Unterstützung als NX (No eXecute) Bit.
Java	Eine von Sun Microsystems forcierte Programmtechnologie. Java-Programme werden nicht wie klassische C/C++ Programme in Plattformabhängige Binaries kompiliert sondern in den so genannten Bytecode. Dieser wird dann von der Java Virtual Machine interpretiert und zur Laufzeit optimiert. Java-Programme können somit auf jeder Plattform ausgeführt werden, für die eine Java Virtual Machine existiert.
JVM	Die Java Virtual Machine ist ein Interpreter für Java Bytecode. Die JVM ist dabei das Bindeglied zwischen Betriebssystem und den plattformunabhängigen Java Anwendungen.
Synchronisierung	Allgemeine Bezeichnung für die Überwachung von konkurrierenden Zugriffen.
Thread	Ein leichtgewichtiger Prozess. Ein Thread teilt den Adressraum mit dem Prozess zu dem er gehört. Dadurch werden einerseits die Kommunikation und andererseits der Kontextwechsel beschleunigt.

12. Verzeichnisse

12.1. Tabellenverzeichnis

Tabelle 1 Referenzierte Dokumente ... 8
Tabelle 2 Abkürzungen ... 8
Tabelle 3 Links ... 9
Tabelle 4 Geplanter Test-Umfang gemäss SDD ([2]) ... 11
Tabelle 5 Hardware-Eckdaten ... 12
Tabelle 6 Software-Umgebung ... 13
Tabelle 7 Software Konfigurationsanpassung ... 14
Tabelle 8 Relevante Performance Indikatoren ... 27
Tabelle 9 Nicht relevante Performance Indikatoren ... 27
Tabelle 10 Profiling- und Testtools ... 28
Tabelle 11 Grad der Skalierung – 1CPU/2CPU (Berechnungszeit) ... 31
Tabelle 12 Grad der Skalierung 1CPU/2CPU (CPU-Zeit) ... 32
Tabelle 13 Thread Mapping ... 33
Tabelle 14 Priority Mapping Java-/Win32 Thread (klein) ... 34
Tabelle 15 Priority Mapping Java-/Win32 Thread (gross) ... 35
Tabelle 16 Skalierung mit variabler Process Priority Class (Berechnungszeit) ... 37
Tabelle 17 Skalierung mit variabler Process Priority Class (CPU-Zeit) ... 38
Tabelle 18 Skalierung mit variabler Win32-Priorität (Berechnungszeit) ... 39
Tabelle 19 Skalierung mit variabler Laststufe (Berechnungszeit) ... 40
Tabelle 20 Skalierung mit variabler Win32-Priorität (CPU-Zeit) ... 41
Tabelle 21 Java Anwendung ohne Affinität ... 42
Tabelle 22 Java Anwendung - Affinität auf CPU 1 ... 42
Tabelle 23 Calc ohne Affinität; Java Anwendung - Affinität auf CPU 1 ... 43
Tabelle 24 Calc - Affinität CPU 0; Java Anwendung - Affinität auf CPU1 ... 43
Tabelle 25 Skalierung 1 CPU ohne Synchronisation ... 45
Tabelle 26 Skalierung 2 CPU ohne Synchronisation ... 46
Tabelle 27 Skalierung 1 CPU mit Methodensynchronisation ... 48
Tabelle 28 Skalierung 1 CPU mit Objektsynchronisation ... 49
Tabelle 29 Skalierung 1 CPU mit CAS-Synchronisation ... 50
Tabelle 30 Skalierung 2 CPU mit Methodensynchronisation ... 52
Tabelle 31 Skalierung 2 CPU mit Objektsynchronisation ... 53
Tabelle 32 Skalierung 2 CPU mit CAS-Synchronisation ... 54
Tabelle 33 Skalierung JOMP-Threads ... 57
Tabelle 34 Glossar ... 59

12.2. Abbildungsverzeichnis

Abbildung 1 Methoden-Synchronisation 1 CPU / 2CPU-Architektur 4
Abbildung 2 Hardware Testplattform 12
Abbildung 3 CodeAnalyst for Windows (AMD) 29
Abbildung 4 ProcessExplorer (Sysinternals) 30
Abbildung 5 Grad der Skalierung – 1CPU/2CPU (Berechnungszeit) 31
Abbildung 6 Grad der Skalierung 1CPU/2CPU (CPU-Zeit) 32
Abbildung 7 Priority Mapping Java-/Win32 Thread (klein) 34
Abbildung 8 Priority Mapping Java-/Win32 Thread (gross) 36
Abbildung 9 Skalierung mit variabler Process Priority Class (Berechnungszeit) 37
Abbildung 10 Skalierung mit variabler Process Priority Class (CPU-Zeit) 38
Abbildung 11 Skalierung mit variabler Win32-Priorität (Berechnungszeit) 39
Abbildung 12 Skalierung mit variabler Laststufe (Berechnungszeit) 40
Abbildung 13 Skalierung mit variabler Win32-Priorität (CPU-Zeit) 41
Abbildung 14 Java Anwendung ohne Affinität 42
Abbildung 15 Java Anwendung - Affinität auf CPU 1 42
Abbildung 16 Calc ohne Affinität; Java Anwendung - Affinität auf CPU 1 43
Abbildung 17 Calc - Affinität CPU 0; Java Anwendung - Affinität auf CPU1 43
Abbildung 18 Skalierung 1 CPU ohne Synchronisation 45
Abbildung 19 Skalierung 2 CPU ohne Synchronisation 46
Abbildung 20 Skalierung 1 CPU mit Methodensynchronisation 48
Abbildung 21 Skalierung 1 CPU mit Objektsynchronisation 49
Abbildung 22 Skalierung 1 CPU mit CAS-Synchronisation 50
Abbildung 23 Skalierung 2 CPU mit Methodensynchronisation 52
Abbildung 24 Skalierung 2 CPU mit Objektsynchronisation 53
Abbildung 25 Skalierung 2 CPU mit CAS-Synchronisation 54
Abbildung 26 Skalierung 1 CPU/2 CPU mit Methodensynchronisation 56
Abbildung 27 Skalierung JOMP-Threads 57

12.3. Code Listings

Listing 1 Ausgabe der Anwendung 29

12.4. Index

Abkürzungen 8
Affinität 59
ANT 13
API 59
Ausschnitt 15
Betrachtungsbereiche .. 11
Betriebssystem 13
Bildgrösse 15
CAS 59
CPU 59
CPU-Anzahl 16
Definitionen 8
DEP 14, 59
Eclipse 13
Ergebnisse 18, 31

Fixe Grössen **15**
Indikatoren **27**
Iterationen **15**
Java **59**
JOMP **13**
JVM **59**
Links **9**
Locking **16**
Messgrössen **10**
numprocs **14**
Priorität **16**
Protokollieren **27**
Rahmenbedingungen **15**
Referenzen **8**
Runtime **13**
Software **13**
Synchroniisierung **59**
Synchronisation **16**
System Information **12**
Testcases **18**
Test-Plattform **12**
Test-Scope **11**
Testtools **28**
Testverfahren **15**
Thread **59**
Tools **28**
Variable Gössen **16**
Worker **16**