FHZ → FACHHOCHSCHULE ZENTRALSCHWEIZ

HTA → HOCHSCHULE FÜR TECHNIK+ARCHITEKTUR LUZERN

Abteilung Informatik

Java

Thread Skalierung

STD - MacBook Pro

Software Test Document

MacBook Pro Architektur

HTA Horw

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	2006-11-21	Rainer Meier	Final Release

Prüfung und Freigabe

Vorname/Name	Dokumentversion	Status	Datum	Visum
Rainer Meier	1.1	Final	2006-11-21	
Marcel Aregger	1.1	Final	2006-11-21	

1. Management Summary

Nachdem uns Roger Diehl unser Wirtschaftspartner unerwartet ein MacBook Pro mit Core 2 Duo (Dual-Core) Prozessor zur Verfügung gestellt hat, haben wir uns entschlossen die Tests aus dem STD ([4]) auf diesem System in gekürzter Form zu wiederholen. Die Ergebnisse untermauern die Resultate und Schlussfolgerungen aus der Conclusion ([6]). Ausserdem erlaubte uns die Apple Architektur einen Einblick in das Skalierungsverhalten unter Mac OS X.

Die neue Plattform hat uns einerseits die Möglichkeit gegeben die Betriebssysteme Windows XP und Mac OS X zu vergleichen. Andererseits konnten wir mit Windows XP Quervergleiche zur vorab getesteten Dual-Opteron Hardware anstellen.

Zwischen Windows XP und Mac OS X sind im direkten Vergleich nur wenige Unterschiede im Bereich der Skalierung auszumachen. Beide Plattformen erlauben durch Multi-Threaded Programmierung die Verteilung auf mehrere Prozessoren/Prozessorkerne. Unter Windows XP fällt ein überproportionaler Einbruch der Skalierung bei nur einem Thread und zwei CPUs auf. Die Leistung liegt hier deutlich unter der Single-Threaded Leistung mit einem einzelnen Kern (1 CPU-Kern deaktiviert). Hierbei scheint es sich um eine Schwäche der Windows-Umgebung zu handeln da dieser Effekt unter Mac OS X nicht beobachtet werden konnte. Ob dies von Windows bzw. dessen Scheduler oder der JVM verursacht wird, konnte nicht abschliessend geklärt werden. Unter Mac OS X konnte dafür ein anderer negativer Effekt beobachtet werden: Die Leistung bricht hier bei mehr als 220 Threads überproportional ein. Dabei steigt die Systemlast (Kernel) stark an. Dies lässt einen erhöhten Scheduling-Aufwand im Mac OS X Kernel vermuten.

Bei der Synchronisierung unter Windows XP sind keine nennenswerten Unterschiede im Vergleich zur Dual-Opteron Maschine auszumachen. Insbesondere bleibt die extrem ansteigende Berechnungszeit bei der Methodensynchronisation mit nur einer aktiven CPU (CPU-Kern) bestehen. Zwischen Mac OS X und Windows XP gibt es nur geringe Unterschiede. Mac OS X scheint mit der Methodensynchronisation etwas besser klar zu kommen und profitiert daher weniger von feinerem Locking oder CAS-Synchronisation.

2. Inhaltsverzeichnis

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]	STD, Software Test Document (Testsreihe auf Opteron Architektur)
[5]	JavaDev JumpStart, Java Entwicklungsumgebung
[6]	Conclusion

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/
[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

Kurz vor Abschluss der Arbeiten erhielten wir die Möglichkeit, unsere Testreihe auf ein MacBookPro mit Intel Core 2 Duo Prozessor (C2D) auszudehnen. Auf dieser Hardware-Plattform konnten Tests sowohl unter Mac OS X als auch unter Windows XP durchgeführt werden. Für eine detaillierte Auswertung fehlte leider die Zeit, wir waren allerdings in der Lage die wichtigsten Testresultate in diesem Dokument zu erfassen und zu kommentieren.

4.2. Testverfahren

Das Testverfahren, die Rahmenbedingungen und die Durchführung der Testreihe entsprechen den Vorgaben gemäss STD ([4]).

5. Test-Scope

Der Test-Scope entspricht dem Testumfang aus [4] mit kleinen Einschränkungen.

Unter Windows XP konnten wir alle Testreihen durchführen. Einzig Tests in Bezug auf die Abbildung von Prioritäten haben wir ausgelassen, da diese nicht Hardware-abhängig sind und auch das Scheduling-Verhalten gleich ausfallen würde wie in [4].

Unter Mac OS X konnten wir das Prioritäts-Mapping mangels tieferer Kenntnisse des Betriebssystems nicht erstellen. Ebenso entfielen Single-CPU Tests, da keine root-Berechtigungen verfügbar waren und somit keine Kernel-Parameter gesetzt werden konnten. Ausserdem waren auf unserem System die Apple Developer Tools nicht installiert. Gemäss einiger Quellen wäre darin eine Option namens ‚CHUD' enthalten. Diese würde es erlauben im laufenden Betrieb die Anzahl der genutzten CPU (Kerne) zu verändern (über Systemeinstellungen ➔ Prozessor). Die Definition von Affinitätsmasken konnten aufgrund des fehlenden Hilfsprogramms ‚taskset' ebenfalls nicht geprüft werden. Skalierungstests auf MAC OS X wurden aber vollumfänglich durchgeführt.

6. Test-Plattform

Das als Testplattform dienende MacBook Pro wurde uns freundlicherweise von unserem Wirtschaftspartner Roger Diehl für die Dauer von zwei Tagen zur Verfügung gestellt. Die nachfolgende Auflistung zeigt die wichtigsten Eckdaten des Systems.

Zusätzlich liegen der Dokumentation Auszüge aus der Mac OS X Systeminformation unter Mac OS X sowie ein Microsoft System Information 7 Dokument (erstellt unter Windows XP) bei:

6.1. Hardware-Dokumentation

Alle Tests wurden auf einem Apple MacBook Pro durchgeführt.

Abbildung 1 MacBook Pro

Tabelle 4 Hardware Eckdaten

Bezeichnung	Konfiguration
Hersteller	Apple
Modell	MacBook Pro 17“
Anzahl Prozessoren	1
Prozessor Typ	Intel Core 2 Duo, 2 CPU-Kerne
Cache	4MB L2 Cache
Taktrate	2.33GHz
FSB-Speed	667MHz
Speicher	2x1GB DDR2
Chipset	Intel I945GT
Festplatte(n)	1x 120GB Fujitsu MHW2120BH
DVD	1x MATSHITA DVDD-R UJ-857D
Netzwerk	Atheros AR5008 WLAN
	Marvel Yukon 88E8053 PCIe Gigabit Ethernet Controller
Grafik	ATI Mobility Radeon X1600

6.2. Software-Dokumentation

6.3. Windows XP

Für die Testreihe 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 eines Notebooks entsprechen womit auch die Ergebnisse auf ähnlichen Maschinen und im Praxiseinsatz vergleichbar sind.

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

Tabelle 5 Software Testumgebung unter Windows-XP

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 [5].

6.4. Mac OS X

Auch hier wurden keine Systemanpassungen vorgenommen. Mangels root-Berechtigungen war dies auch nicht möglich. Die folgende Tabelle schliesst die wichtigsten Software-Eckdaten ein.

Tabelle 6 Software Testumgebung unter Mac OS X

Bezeichnung	Beschreibung
Mac OS X	Die Tests wurden unter Mac OS X Version 10.4.8 durchgeführt.
Aktivitäts-Anzeige	Die Prozessoraktivität und CPU-Zeit wurde mit der Apple Aktivitäts-Anzeige in der Version 1.5.2 (56) ermittelt.
Java VM	Wir verwendeten für die Tests die Sun HotSpot Java Client VM Version 1.5.0_06-68, mixed mode, sharing.
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 [5].

7. Testcases Prioritäten

7.1. Windows XP

Die Prioritätsabbildung wurde bereits in [4] detailliert analysiert und braucht hier nicht mehr wiederholt zu werden. Stichproben haben bestätigt, dass die Abbildung nicht Architekturabhängig ist und auf dem MacBook Pro genau gleich wie auf der Dual-Opteron Plattform aussieht.

7.2. Mac OS X

Wie einleitend erwähnt haben uns für die Prioritätsanalyse unter Mac OS X sowohl die Zeit als auch das tiefere Wissen der Prioritätsbehandlung durch das Betriebssystem gefehlt. Deshalb haben wir auf diese Testreihe verzichtet.

8. Testcases Affinität

8.1. Windows XP

- Auch hier fällt auf, dass ein einzelner Thread 50%/50% auf beide CPUs verteilt wird. Auch bei C2D.

8.2. Mac OS X

- Mangels root-Berechtigungen konnten wir keine Kernel-Parameter setzen
- Die Apple Developer Tools (Option ‚CHUD') war nicht installiert. Somit war keine Limitierung der CPU-Anzahl möglich.
- Das Tool ‚taskset' zur Definition einer Affinitätsmaske stand auf dem Testsystem nicht zur Verfügung.
- Durch setzen einer Affinitätsmaske auf den ‚init'-Prozess wäre es unter Unix/Mac OS X möglich alle Prozesse auf bestimmte CPUs zu beschränken. Der ‚init'-Prozess ist dabei der Vaterprozess aller Prozesse. Durch Vererbung wird bei der Definition einer Affinitätsmaske auf den ‚init'-Prozess eine Maske für alle Prozesse definiert. Anschliessend können einzelnen Unterprozessen abweichende/komplementäre Affinitätsmasken zugewiesen und somit andere CPUs exklusiv zugeteilt werden.
- Auch Mac OS X verteilt einen einzelnen Thread 50%/50% auf beide vorhandenen Kerne.

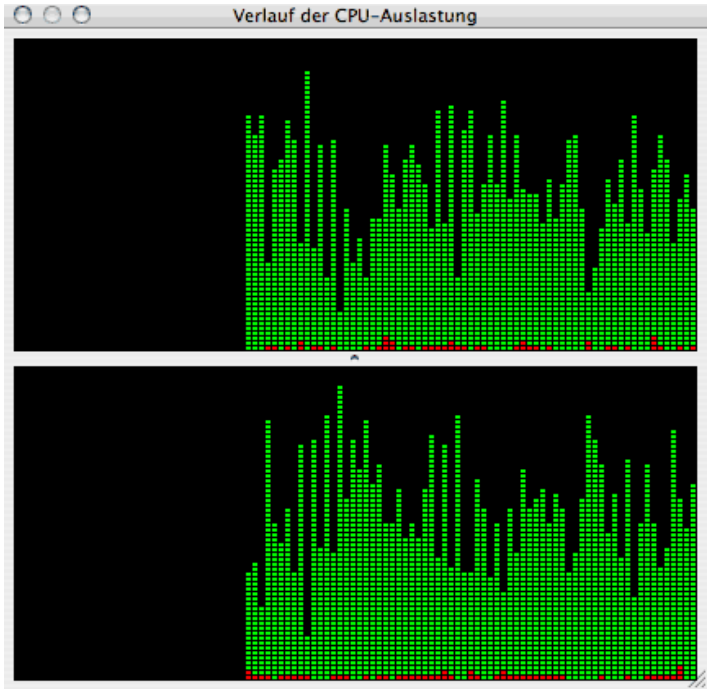

Abbildung 2 CPU-Auslastung, 1 Thread, 2 Kerne, Mac OS X

9. Testcases Skalierung

9.1. Windows XP

9.1.1. Testcase 8

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

9.1.1.1. Skalierung 1 CPU ohne Synchronisation

Tabelle 7 Skalierung 1 CPU ohne Synchronisation Windows XP

1 CPU	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	43.1763	43.5993	43.3400	42.9803	43.0063	43.4493
CPU-Zeit	0:42	0:42	0:43	0:42	0:42	0:43
Faktor Skalierung	1.00	0.99	1.00	1.00	1.00	0.99

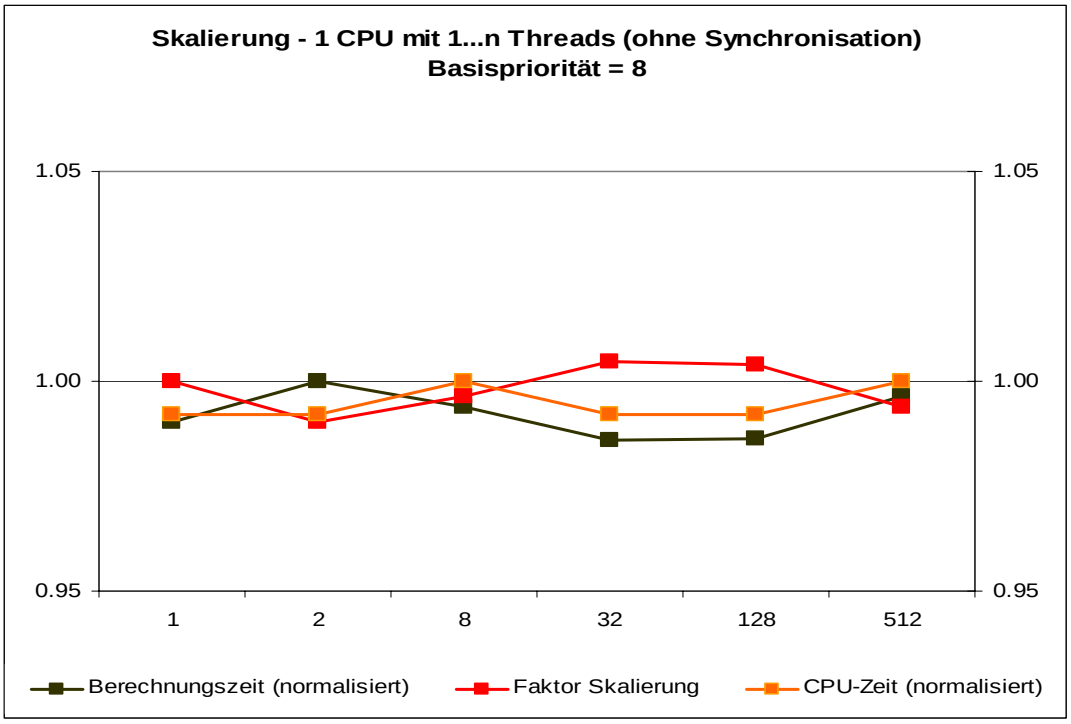

Abbildung 3 Skalierung 1 CPU ohne Synchronisation Windows XP

9.1.1.2. Skalierung 2 CPU ohne Synchronisation

Tabelle 8 Skalierung 2 CPU ohne Synchronisation Windows XP

2 CPU	Anzahl Threads					
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	512
Berechnungszeit	64.9386	21.8330	21.6916	21.3828	21.6316	22.6496
CPU-Zeit	1:06	0:42	0:42	0:42	0:42	0:45
Faktor Skalierung	1.00	2.97	2.99	3.04	3.00	2.87

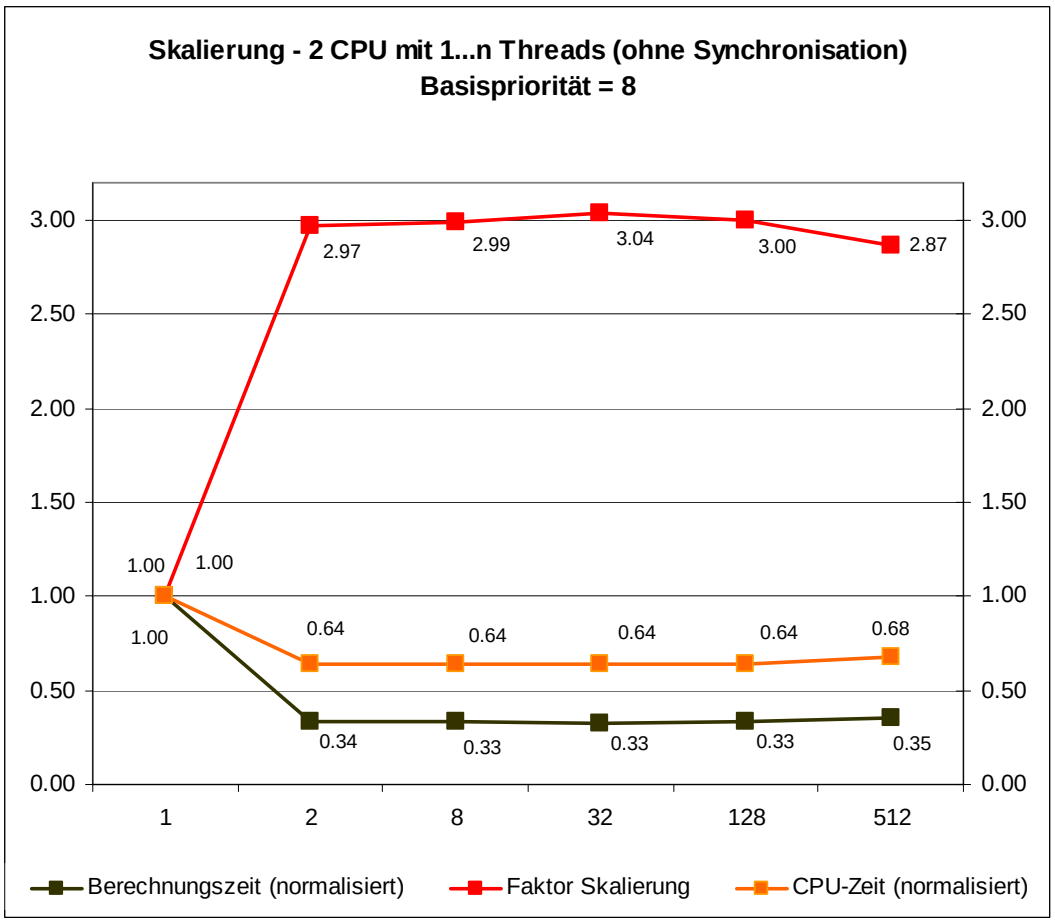

Abbildung 4 Skalierung 2 CPU ohne Synchronisation Windows XP

9.1.1.3. Ergebnisse

- Auf der 1 CPU-Architektur für 1 bis 512 Threads annähernd konstante Berechnungs- und CPU-Zeiten.
- Auf der 2 CPU-Architektur ab 2 Threads durch Verteilung der Threads eine Reduktion um zwei Drittel der Berechnungszeit.
- Auf der 2 CPU-Architektur wird ab 2 Threads durch Verteilung der Threads ein Skalierungsfaktor von nahezu 3 erreicht
- Auf der 2 CPU Architektur ab 2 Threads eine Reduktion der CPU-Zeit um einen Drittel.
- 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

9.1.1.4. Conclusion

Auf der Single-CPU Architektur resultiert im Vergleich zur AMD Opteron Testplattform (siehe [4]) eine Reduktion der Berechnungs- und CPU-Zeit um rund 40%. Der Core 2 Duo Prozessor besitzt mit 2.33GHz eine um 16.5% höhere Taktungt als der 2GHz Opteron Prozessor. Daraus resultiert eine Effizienzsteigerung des C2D im Bezug auf die Rechnungsleistung pro MHz.

Im Dual-CPU Test resultiert ab zwei Threads eine Halbierung der Berechnungszeit. Dies entspricht einem Skalierungsfaktor von 2. Der Test mit nur einem Thread zeigt dabei eine unerwartet hohe Berechnungs- und CPU-Zeit. Beide liegen deutlich über den Ergebnissen des Single-CPU Tests. die Berechnungszeit mit nur einem Thread liegt bei 66 Sekunden während beim Single-CPU Test diese bei erwarteten 43 Sekunden lag.

Der markante Einbruch der Leistung des C2D Prozessors mit einer Single-Threaded Anwendung war ein völlig unerwartetes Systemverhalten. Dabei scheint es sich um eine Schwäche von Windows XP auf der C2D Architektur zu handeln. Die Tests unter Mac OS X zeigten keine derartige Schwäche (siehe Kapitel 9.2). Ebenso zeigten die Tests auf dem Dual-Opteron System kein vergleichbares Systemverhalten (siehe [4]).

9.1.2. Testcase 9

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

9.1.2.1. Skalierung 1 CPU - Methodensynchronisation

Tabelle 9 Skalierung 1 CPU mit Methodensynchronisation Windows XP

| 1 CPU | Methodensynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	42.6690	43.2187	80.2963	81.6973	78.8650	80.8003
CPU-Zeit	0:42	0:42	1:18	1:20	1:17	1:19
Faktor Skalierung	1.00	0.99	0.53	0.52	0.54	0.53

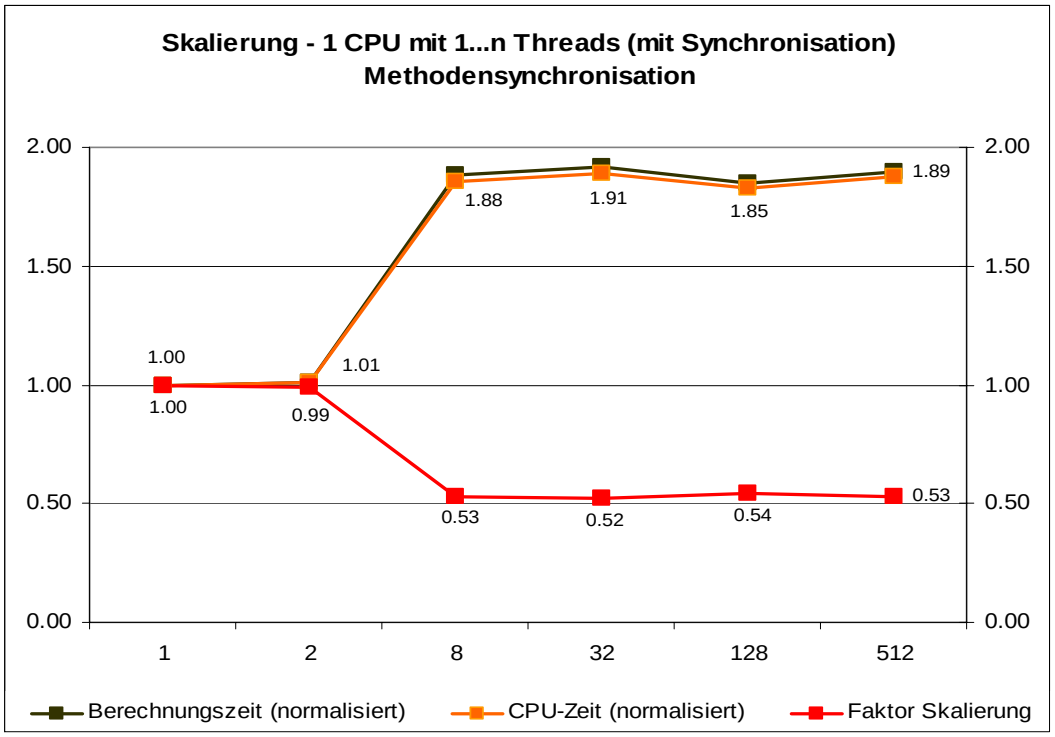

Abbildung 5 Skalierung 1 CPU mit Methodensynchronisation Windows XP

9.1.2.2. Skalierung 1 CPU - Objektsynchronisation

Tabelle 10 Skalierung 1 CPU mit Objektsynchronisation Windows XP

| 1 CPU | Objektsynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	42.7633	43.0817	43.2010	42.7633	43.1087	43.4597
CPU-Zeit	0:42	0:43	0:43	0:42	0:43	0:43
Faktor Skalierung	1.00	0.99	0.99	1.00	0.99	0.98

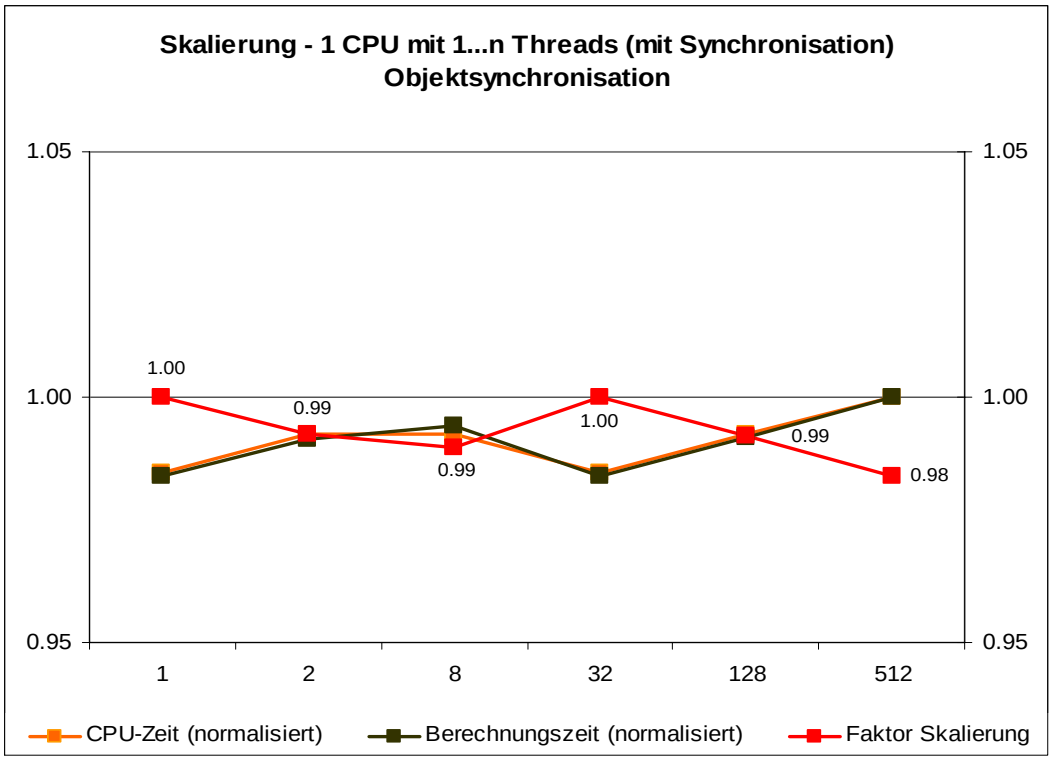

Abbildung 6 Skalierung 1 CPU mit Objektsynchronisation Windows XP

9.1.2.3. Skalierung 1 CPU - CAS-Synchronisation

Tabelle 11 Skalierung 1 CPU mit CAS-Synchronisation Windows XP

| 1 CPU | CAS	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	42.7713	43.0937	43.2083	42.7220	42.8483	43.4240
CPU-Zeit	0:42	0:43	0:43	0:42	0:42	0:43
Faktor Skalierung	1.00	0.99	0.99	1.00	1.00	0.98

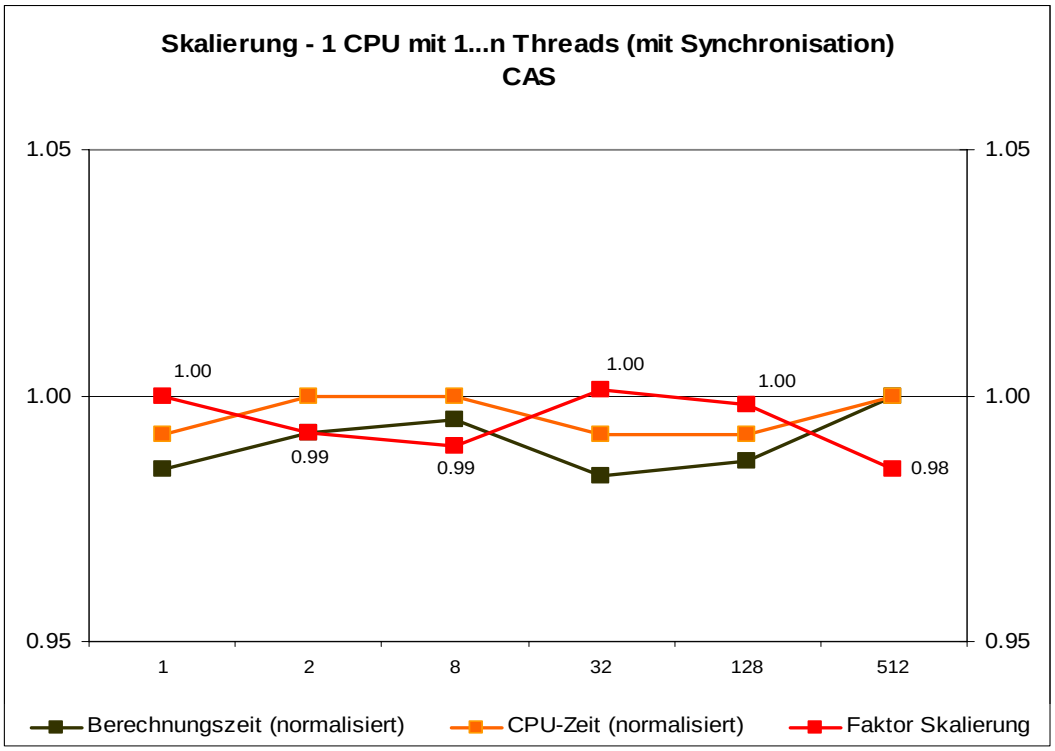

Abbildung 7 Skalierung 1 CPU mit CAS-Synchronisation Windows XP

9.1.2.4. 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 mit minimalen Schwankungen (Messungenauigkeit).

CAS-Synchronisation

- Für CAS-Synchronisation mit 1 bis 512 Threads annähernd konstante Berechnungs- und CPU-Zeit mit minimalen Schwankungen (Messungenauigkeit).

9.1.2.5. Conclusion

Das C2D System bestätigt das gemessene Verhalten der Dual-Opteron Architektur ([4]) im Bezug auf die Methodensynchronisation in Single-CPU Konfiguration. Das bedeutet, dass ein markanter Anstieg der Berechnungs- und CPU-Zeit mit mehr als 2 Threads zu verzeichnen ist.

Durch den Einsatz von Objekt- oder CAS-Synchronisation lässt sich eine lineare Skalierung für 1 bis 512 Threads wiederherstellen.

Siehe dazu auch [6].

9.1.2.6. Skalierung 2 CPU - Methodensynchronisation

Tabelle 12 Skalierung 2 CPU mit Methodensynchronisation Windows XP

| 2 CPU | Methodensynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	68.4903	22.0663	24.9543	26.3010	26.4487	28.0297
CPU-Zeit	1:09	0:43	0:49	0:52	0:52	0:56
Faktor Skalierung	1.00	3.10	2.74	2.60	2.59	2.44

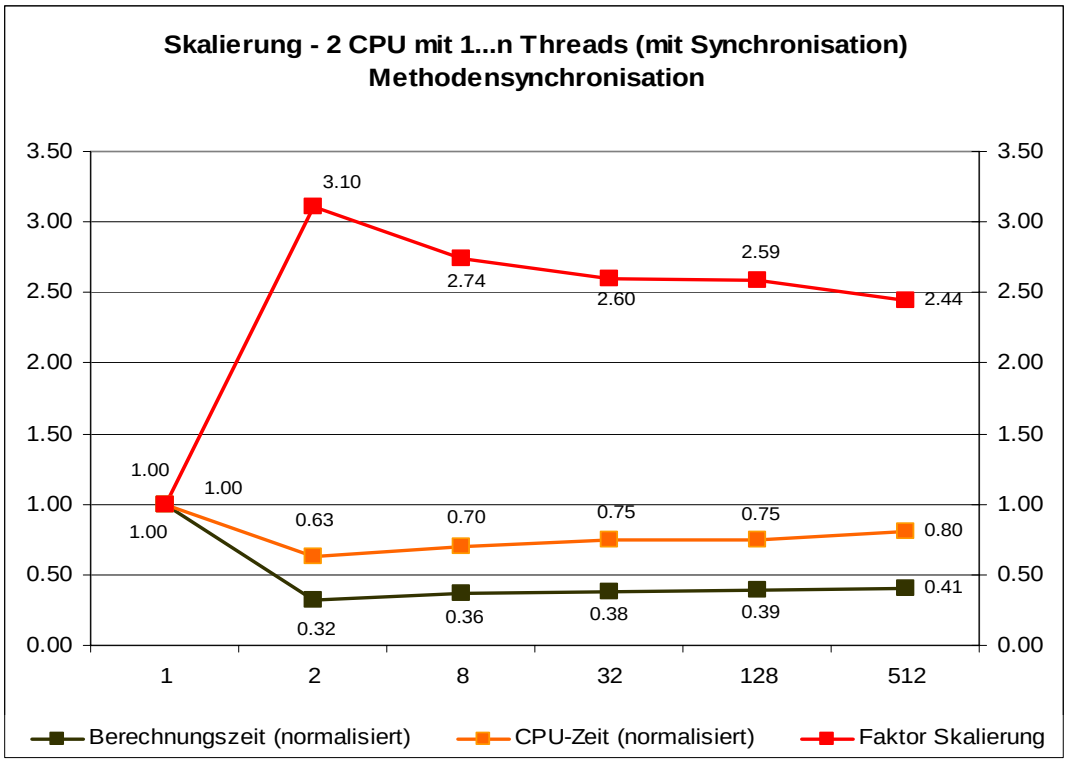

Abbildung 8 Skalierung 2 CPU mit Methodensynchronisation Windows XP

9.1.2.7. Skalierung 2 CPU - Objektsynchronisation

Tabelle 13 Skalierung 2 CPU mit Objektsynchronisation Windows XP

| 2 CPU | Objektsynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	56.5320	22.0180	24.9330	26.1983	25.7687	27.4867
CPU-Zeit	0:58	0:44	0:49	0:52	0:51	0:55
Faktor Skalierung	1.00	2.57	2.27	2.16	2.19	2.06

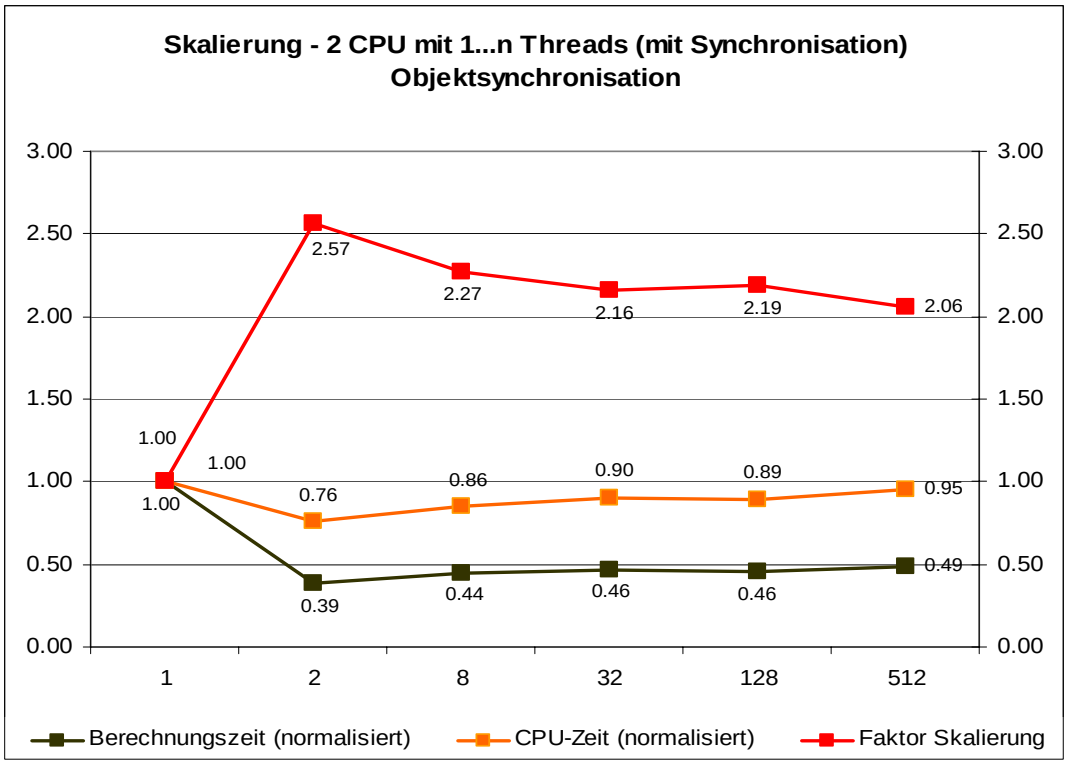

Abbildung 9 Skalierung 2 CPU mit Objektsynchronisation Windows XP

9.1.2.8. Skalierung 2 CPU - CAS-Synchronisation

Tabelle 14 Skalierung 2 CPU mit CAS-Synchronisation Windows XP

| 2 CPU | CAS	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	60.7953	21.8513	21.7180	21.6490	21.7983	22.8867
CPU-Zeit	1:02	0:43	0:43	0:43	0:44	0:46
Faktor Skalierung	1.00	2.78	2.80	2.81	2.79	2.66

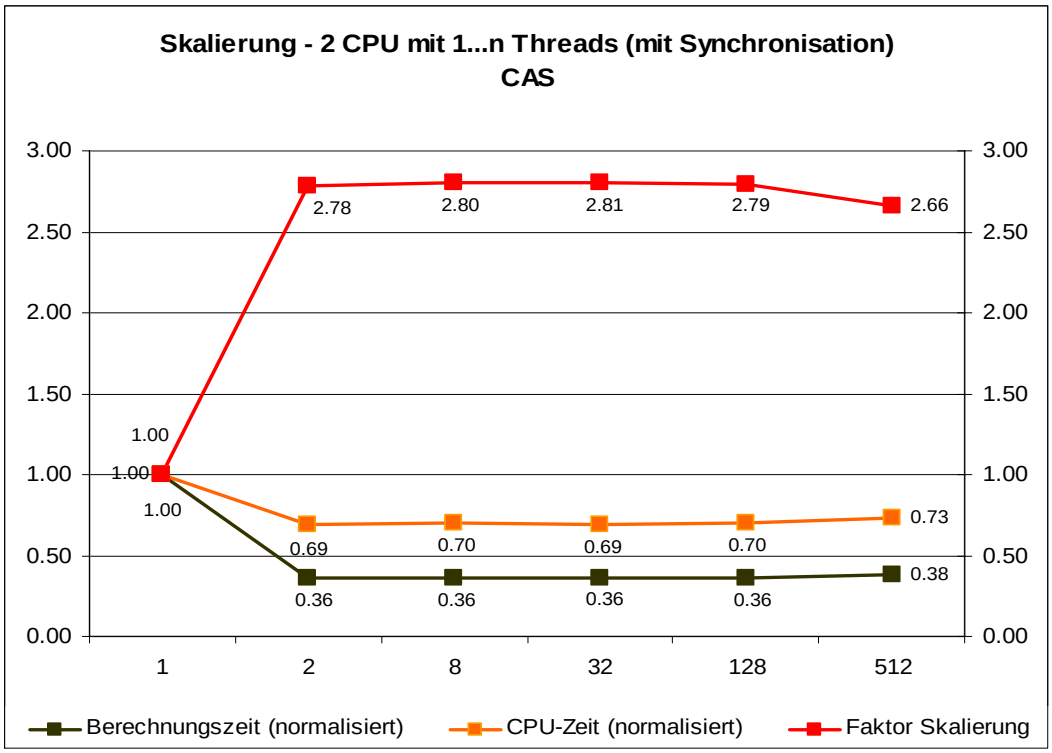

Abbildung 10 Skalierung 2 CPU mit CAS-Synchronisation Windows XP

9.1.2.9. 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 und 512 Threads ergibt sich eine moderate Zunahme von CPU- und Berechnungs-Zeit

Objektsynchronisation

- Für Objektsynchronisation ähnlicher Verlauf im Vergleich zur Methoden Synchronisation
- Der Anstieg der Berechnungs- und CPU-Zeit zwischen 3 und 512 Threads ist im Vergleich zur Methodensynchronisation flacher.

CAS-Synchronisation

- Für CAS-Synchronisation resultiert ein ähnlicher Verlauf im Vergleich zur Objektsynchronisation
- Für CAS-Synchronisation annähernd linearer Verlauf der CPU- und Berechnungs-Zeit.

Direkter Vergleich 1 CPU und 2CPU mit Methodensynchronisation

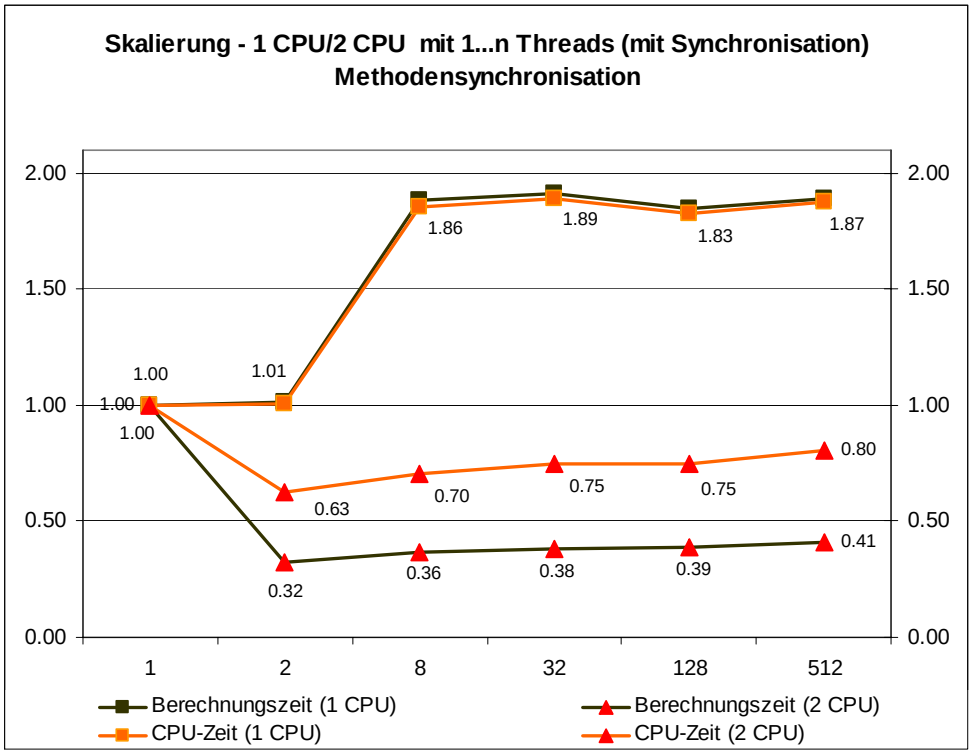

Abbildung 11 Skalierung 1 CPU/2 CPU mit Methodensynchronisation

9.1.2.10. Conclusion

Durch den Einsatz von Objekt- und CAS-Synchronisation lässt sich der Skalierungs-Faktor bei einer hohen Thread-Anzahl deutlich verbessern. Das plattformübergreifende Verhalten zeigt, dass der Synchronisationsaufwand nur vom Synchronisationsverfahren und nicht von der verwendeten Plattform abhängig ist.

Siehe auch [6].

9.2. Mac OS X

9.2.1. Testcase 8

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

9.2.1.1. Skalierung 2 CPU ohne Synchronisation

Tabelle 15 Skalierung 2 CPU ohne Synchronisation MAC OS X

2 CPU	Anzahl Threads							
JThreadpriorität = 5, Basispriorität = 8	1	2	8	32	128	220	250	512
Berechnungszeit	42.7966	23.6312	26.7486	23.7568	22.9528	25.6498	45.8306	47.1080
CPU-Zeit	0:44	0:45	0:48	0:45	0:44	0:50	1:29	1:32
Faktor Skalierung	1.00	1.81	1.60	1.80	1.86	1.67	0.93	0.91

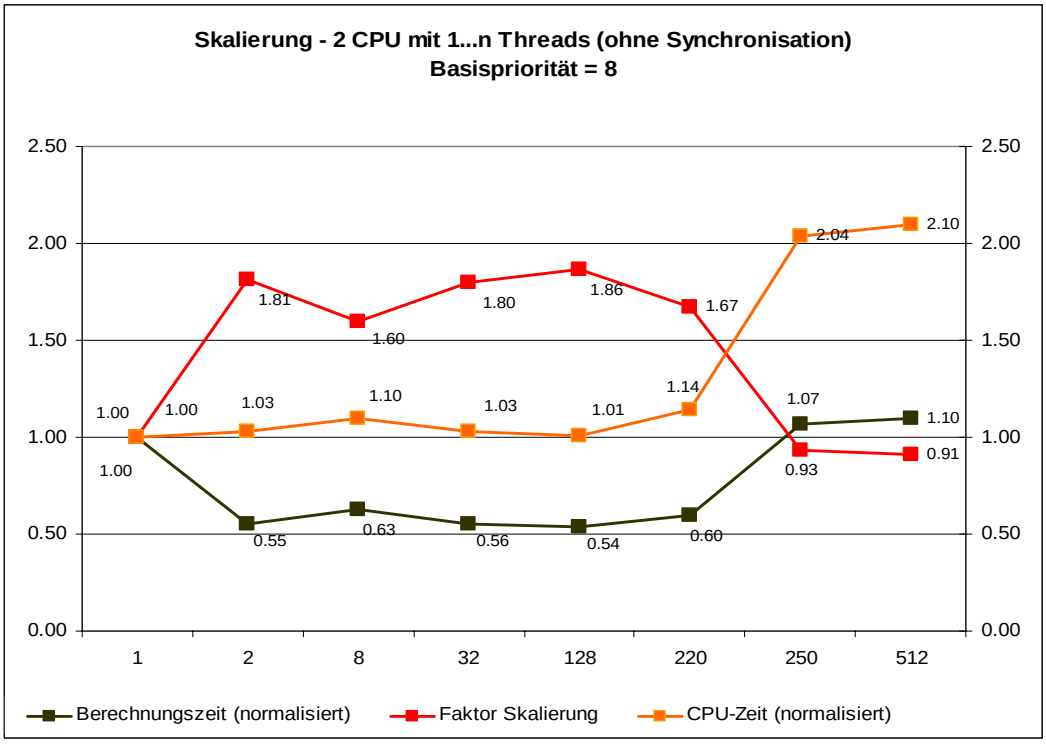

Abbildung 12 Skalierung 2 CPU ohne Synchronisation MAC OS X

9.2.1.2. Ergebnisse

- Auf der 2 CPU-Architektur ab 2 Threads durch Verteilung der Threads eine Reduktion um die Hälfte der Berechnungszeit.
- Auf der 2 CPU-Architektur wird ab 2 Threads durch Verteilung der Threads ein Skalierungsfaktor von nahezu 2 erreicht
- Die CPU-Zeit bleibt annähernd konstant für 2 bis ca. 220 Threads
- Markanter Anstieg von CPU- und Berechnungs-Zeit mit mehr als 220 Threads (ca. Faktor 2). Dadurch resultiert ein Einbruch der Skalierung.

9.2.1.3. Conclusion

Wir haben festgestellt, dass bei 512 Threads die CPU- und Berechnungs-Zeit im Vergleich zu 128 Threads sprunghaft ansteigt. Durch iteratives reduzieren der Thread-Anzahl haben wir den Schwellwert bei ca. 220 Threads lokalisiert. Die Suche nach dem Schwellwert hat gezeigt, dass dieser nicht immer exakt bei 220 Threads liegt und somit wahrscheinlich von weiteren Systemparametern abhängig ist. Bei Überschreitung des Grenzwertes zeigte die Aktivitäts-Anzeige (System-Tool) eine erhöhte System-Aktivität (Kernel). Dieses Verhalten scheint charakteristisch für Mac OS X zu sein. Unter Windows XP konnte ein derartiges Verhalten nicht beobachtet werden (siehe 9.1).

Im Vergleich zu Windows XP fällt auf, dass ein einzelner Thread auf der 2-CPU Architektur weitaus effizienter abgearbeitet wird. Der unter 9.1.1.4 beschriebene überproportionale Einbruch zwischen 1 und 2 Threads war hier nicht zu beobachten.

9.2.2. Testcase 9

9.2.2.1. Skalierung 2 CPU - Methodensynchronisation

Tabelle 16 Skalierung 2 CPU mit Methodensynchronisation MAC OS X

| 2 CPU | Methodensynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	42.8380	24.4330	26.6940	23.6673	22.9620	46.3147
CPU-Zeit	0:44	0:45	0:48	0:45	0:44	1:30
Faktor Skalierung	1.00	1.75	1.60	1.81	1.87	0.92

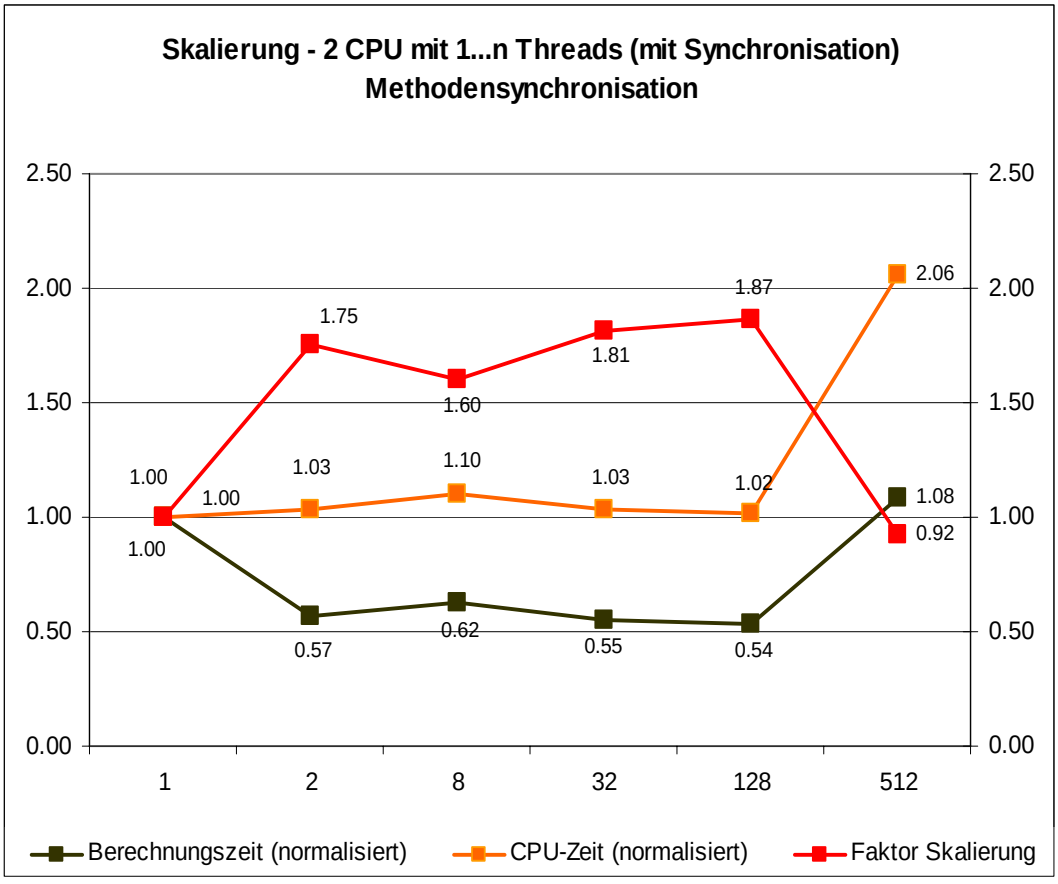

Abbildung 13 Skalierung 2 CPU mit Methodensynchronisation MAC OS X

9.2.2.2. Skalierung 2 CPU - Objektsynchronisation

Tabelle 17 Skalierung 2 CPU mit Objektsynchronisation MAC OS X

| 2 CPU | Objektsynchronisation	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	42.9367	24.0217	26.4450	25.8353	22.6377	46.0590
CPU-Zeit	0:44	0:45	0:48	0:45	0:44	1:30
Faktor Skalierung	1.00	1.79	1.62	1.66	1.90	0.93

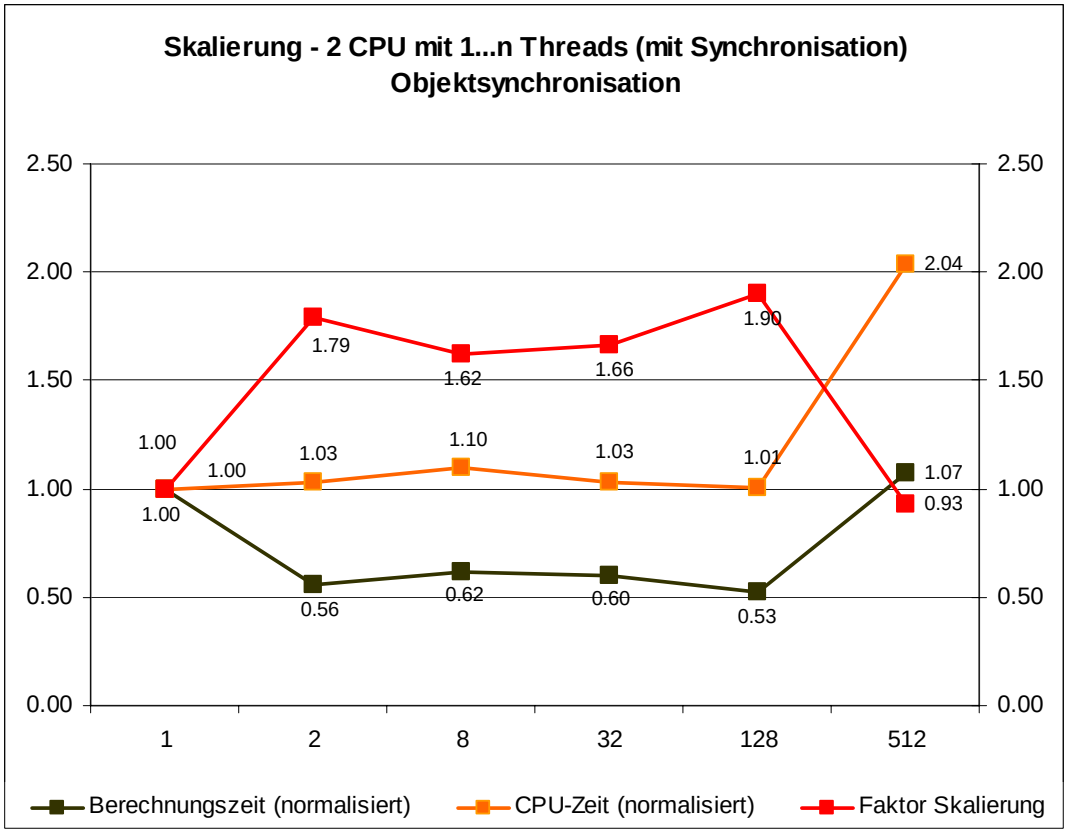

Abbildung 14 Skalierung 2 CPU mit Objektsynchronisation MAC OS X

9.2.2.3. Skalierung 2 CPU - CAS-Synchronisation

Tabelle 18 Skalierung 2 CPU mit CAS-Synchronisation MAC OS X

| 2 CPU | CAS	Anzahl Threads					
JThreadpriorität = 5, Basis-priorität = 8	1	2	8	32	128	512
Berechnungszeit	42.9013	23.8553	26.5427	23.6820	22.6637	44.6730
CPU-Zeit	0:44	0:45	0:49	0:45	0:44	1:28
Faktor Skalierung	1.00	1.80	1.62	1.81	1.89	0.96

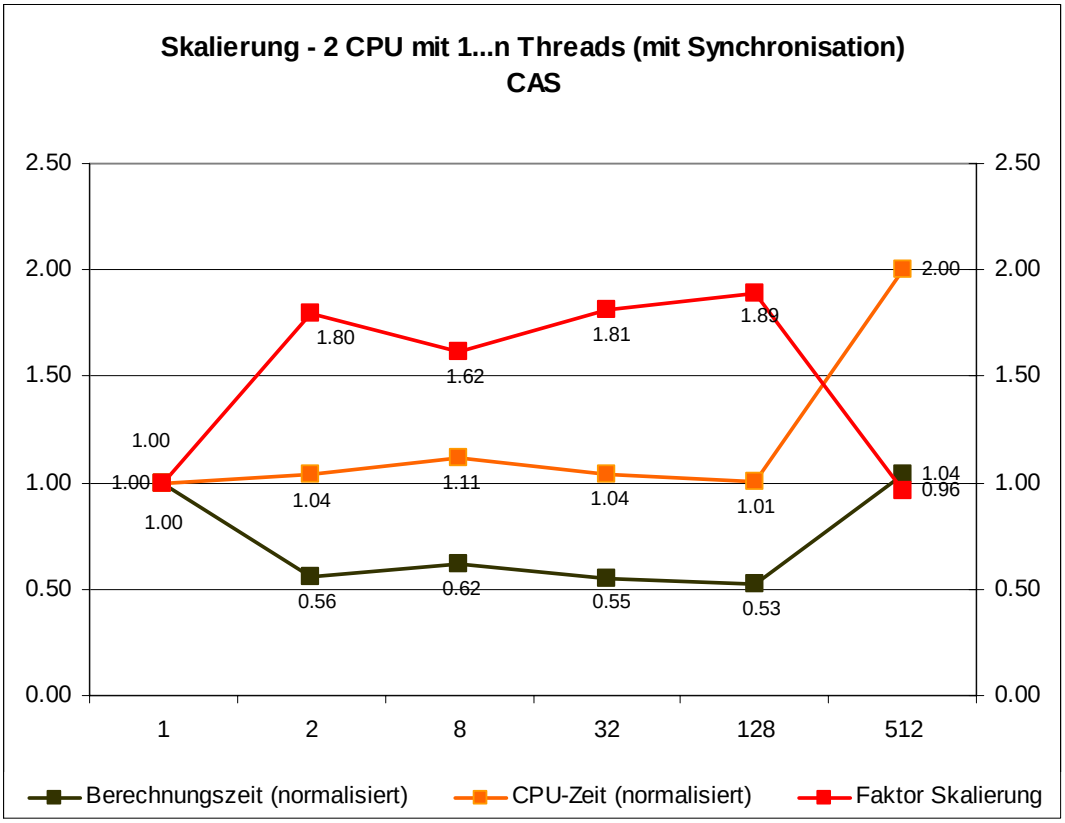

Abbildung 15 Skalierung 2 CPU mit CAS-Synchronisation MAC OS X

9.2.2.4. Ergebnisse

- Zwischen 1 und 128 Threads linearer Verlauf der CPU- und Berechnungs-Zeit (mit Ausnahme es Tests mit 8 Threads)
- Bei 8 Threads ist eine messbare Zunahme der CPU- und Berechnungszeit für alle Synchronisationsmethoden
- Ab 128 Threads ist ein markanter Anstieg von CPU- und Berechnungszeit zu verzeichnen (siehe auch Beschreibung in 9.2.1)

9.2.2.5. Conclusion

Der markante Anstieg der CPU- und Berechnungszeit bei mehr als 128 Threads ist unabhängig davon, ob Synchronisation eingesetzt wird oder nicht (vergleiche Dazu die Messwerte aus 9.2.1).

Das Systemverhalten mit 8 Threads ist bei allen Synchronisationsmethoden registriert worden. Eine eindeutige Erklärung dafür konnte aber nicht gefunden werden.

Der direkte Vergleich der Synchronisationsmechanismen unter Mac OS X zeigt keinen eindeutigen Sieger. Dem gegenüber steht Windows XP wo verschiedene Synchronisationsmethoden deutliche Unterschiede im Skalierungsverhalten aufzeigen (siehe 9.1).

10. Verzeichnisse

10.1. Tabellenverzeichnis

10.2. Abbildungsverzeichnis

10.3. Index