Praktikum angewandte Systemsoftwaretechnik: Vergleich nichtblockierender Queues

Michael Banken Lorenz Haspel

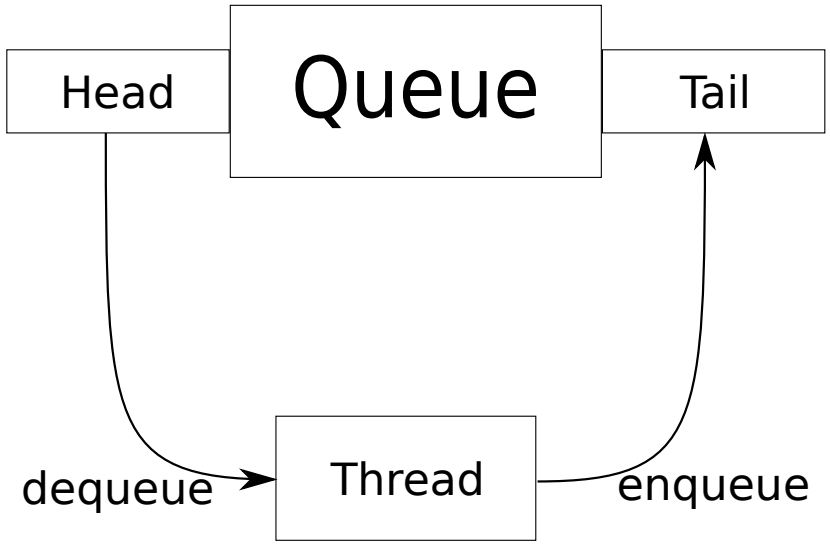
Friedrich-Alexander Universität Erlangen-Nürnberg

20.9.2013

Projektübersicht

- im Rahmen von LAOS/OctoPOS
- Queues als zentrale Datenstrukur für die Verwaltung von Threads
- paralleler Zugriff auf gemeinsame Daten
- unsere Zielsetzung: Performanzanalyse und die Entwicklung paralleler Tests mit verschiedenen nichtblockierenden Queues

Durchgeführte Tests: "Roundtest"

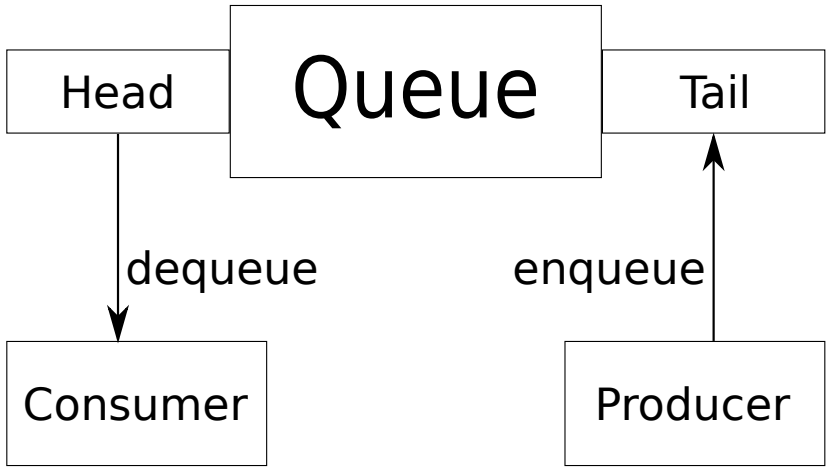


- eine Queue, zu Beginn gefüllt
- Threads entfernen je ein Element aus der Queue und fügen dieses wieder ein.
- Elemente bleiben konstant, werden immer wieder verwendet

Durchgeführte Tests: "Roundtest"

- Versuchsreihen mit variierender Anzahl an Threads und Elementen
- feste Gesamtzahl an Operationen
- Zeit wird gemessen
- drei Testreihen:
  - halb so viele Elemente wie Threads
  - genau so viele Elemente wie Threads
  - doppelt so viele Elemente wie Threads

Durchgeführte Tests: "MPMC-Test"



- Multiple Producers Multiple Consumers
- Queue zu Beginn leer
- Produzententhreads fügen Elemente ein
- Konsumententhreads nehmen Elemente raus
- jedes Element wird nur einmal verwendet

Durchgeführte Tests: "MPMC-Test"

- Versuchsreihe mit variierender Anzahl an Threads
- feste Anzahl an Elementen, die die Queue durchlaufen
- zwei Testreihen:
  - variable Zahl an Produzenten, ein Konsument
  - gleiche Zahl von Produzenten wie Konsumenten

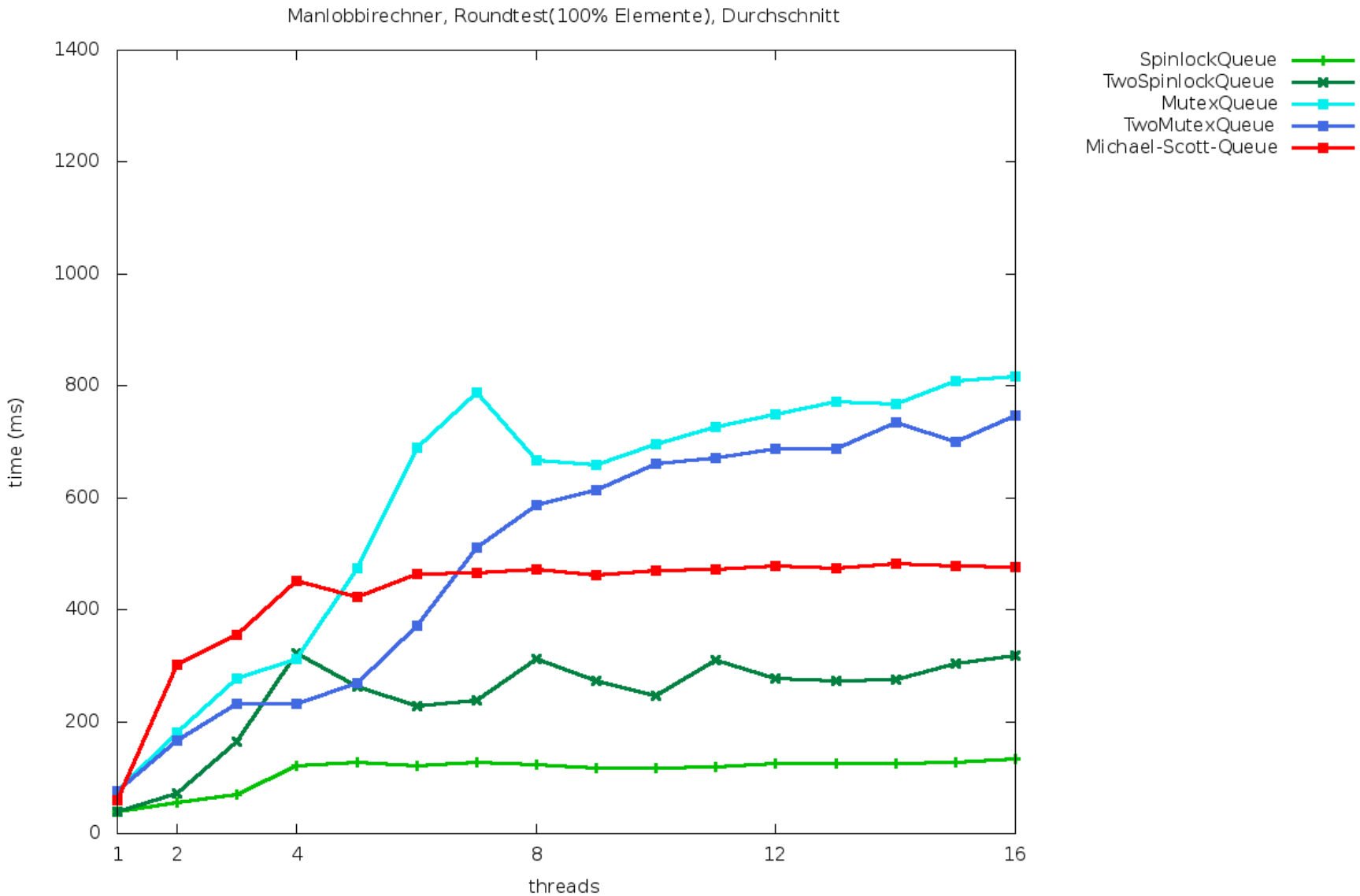
Verwendete Architekturen

- Core i7 aus der Manlobbi:
  - 4 Kerne, 8 durch Hyperthreading
  - Tests neben normaler Betriebslast (Programme andere User, X-Server, Ubuntu-Hintergrundprogramme)
- Haswellrechner "Fastbox":
  - 8 Kerne, Intel-Haswell Prozessoren
  - Test von Queues mit naiver Implementierung von Haswell-Transaktionen
  - atomare Transaktionen auf Prozessorebene
- 48-Kern-Rechner "Bigbox":
  - Tests bei massiv parallelem Zugriff

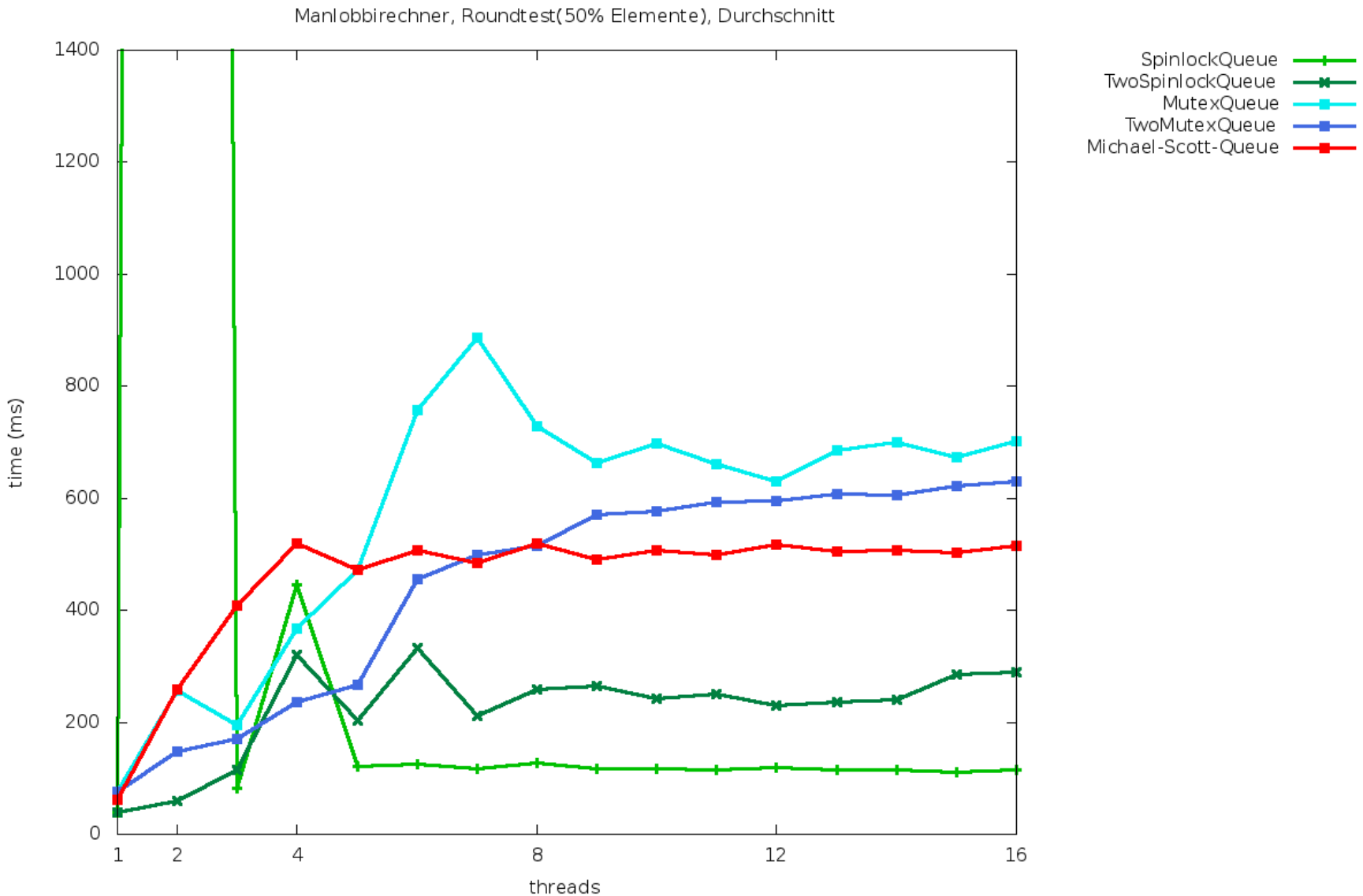
Verwendete Queues

- SimpleQueue: Queue mit einem globalen Lock für en- und dequeue-Operationen
  - Varianten: Spinlock, Mutex, Haswell-Transaktion
- TwoLockQueue: Queue mit zwei Locks, je eines für en- und dequeue Operationen
  - Varianten: Spinlock, Mutex, Haswell-Transaktionen
- Michael-Scott-Queue: nicht-blockierende Queue
  - verwendet CAS-Operationen
- MPSC-Queue: Multiple-Producers-Single-Consumer-Queue
  - spezialisierte nicht-blockierende Queue

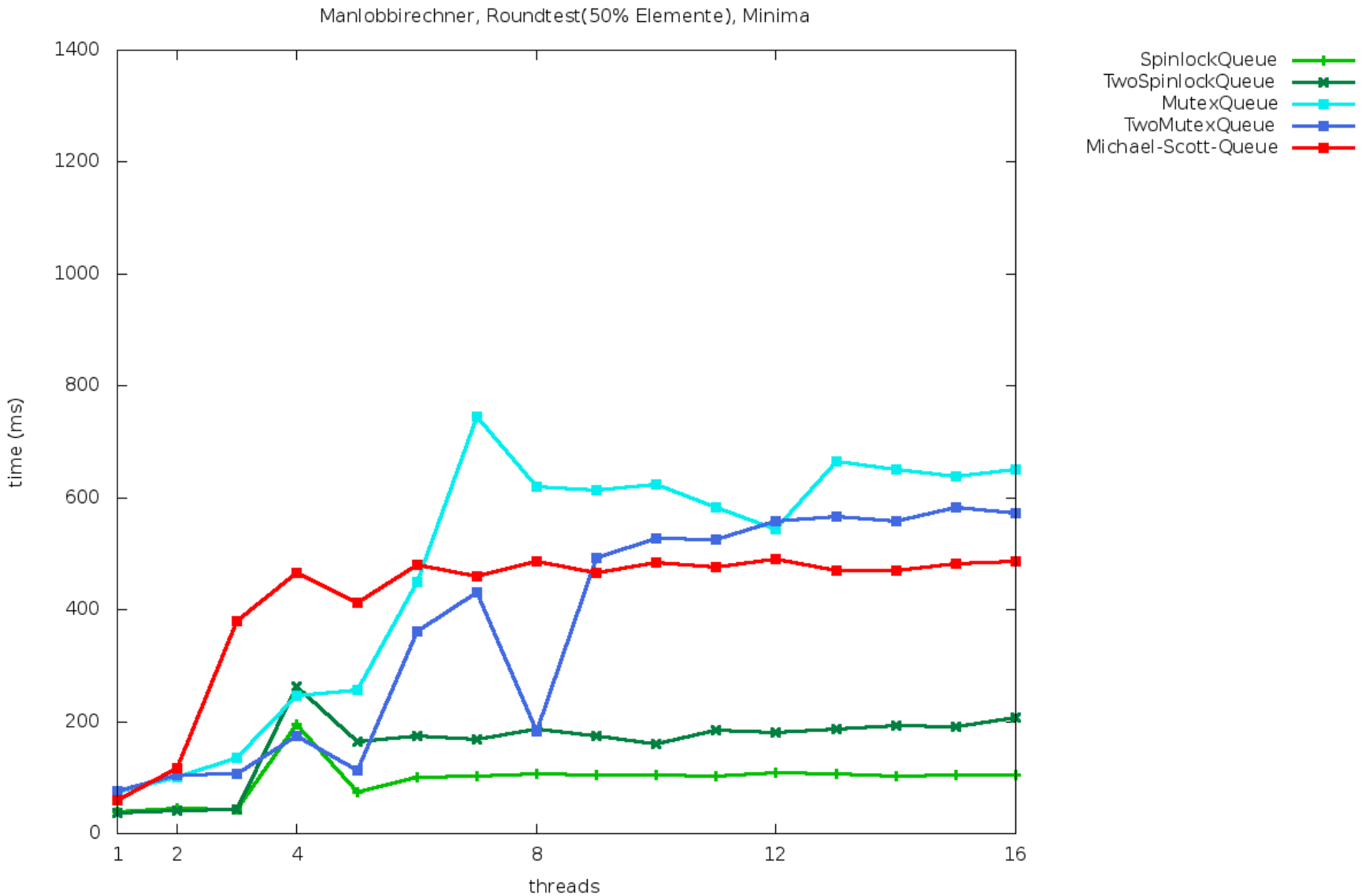
Ergebnisse


Manlobbirechner, Roundtest(100% Elemente), Durchschnitt
time (ms)
threads
0
200
400
600
800
1000
1200
1400
1
2
4
8
12
16
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue

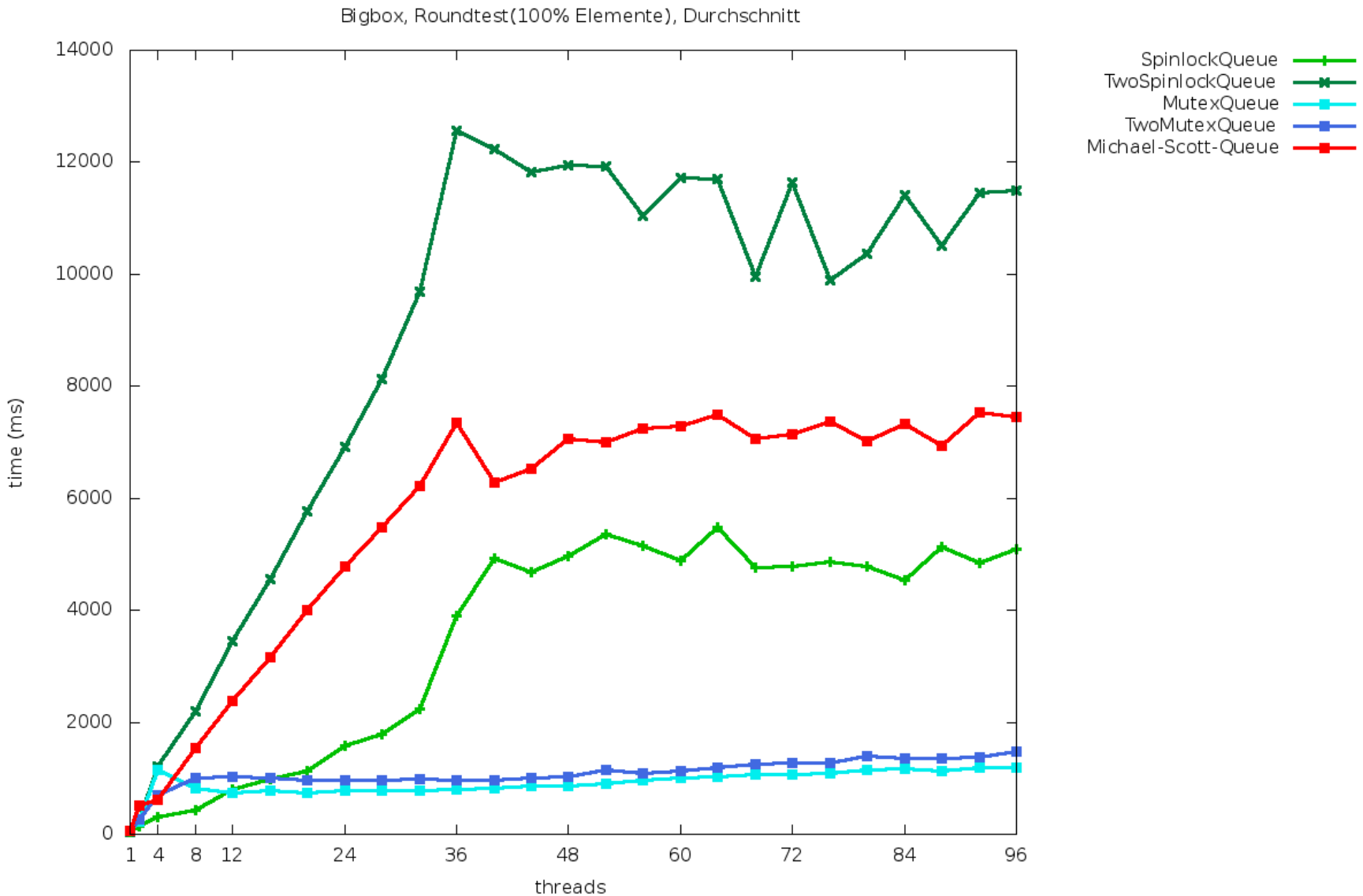
Ergebnisse


Manlobbirechner, Roundtest(50% Elemente), Durchschnitt
time (ms)
threads
0
200
400
600
800
1000
1200
1400
1
2
4
8
12
16
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue

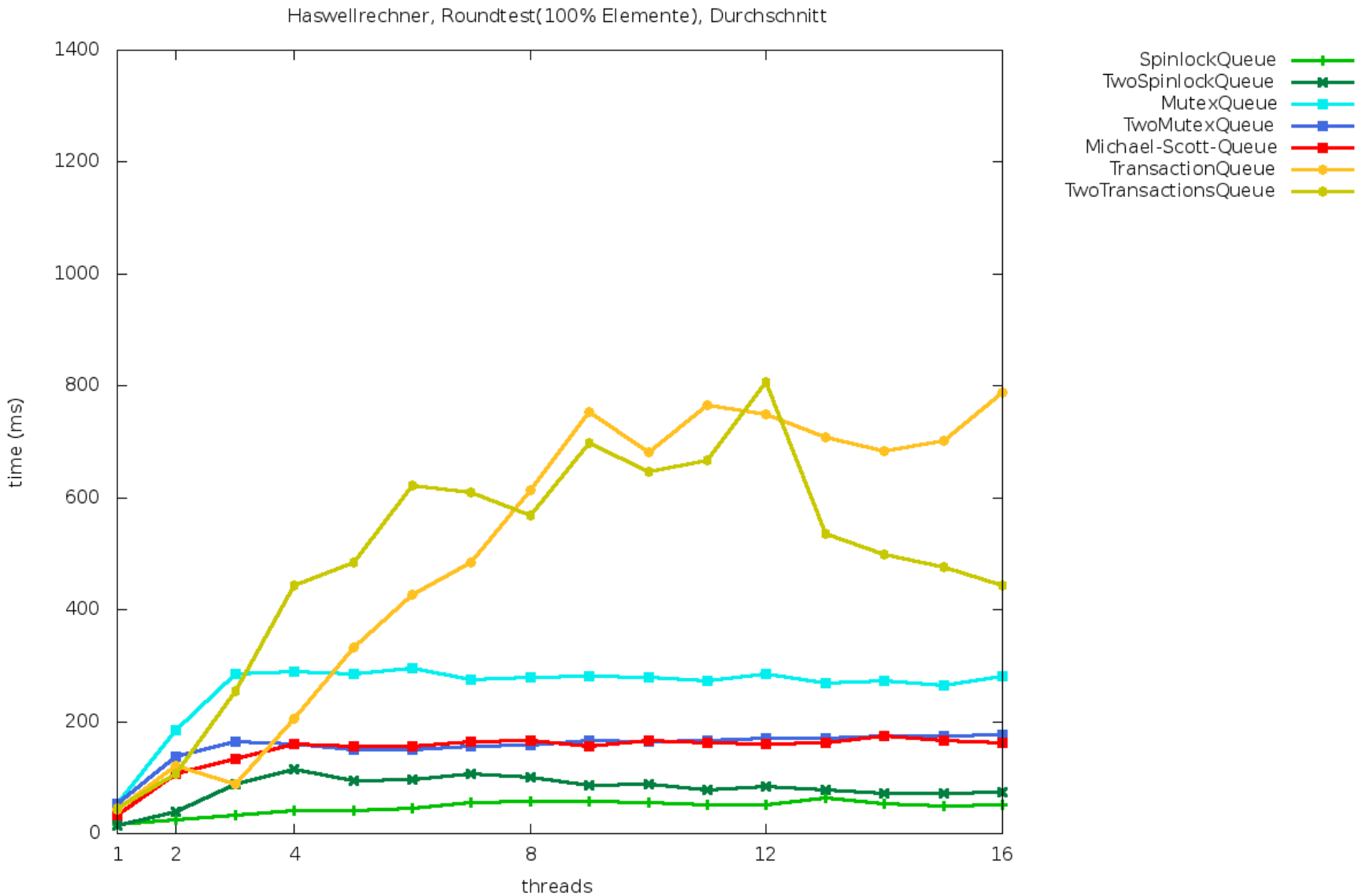
Ergebnisse


Manlobbirechner, Roundtest(50% Elemente), Minima
time (ms)
threads
0
200
400
600
800
1000
1200
1400
1
2
4
8
12
16
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue

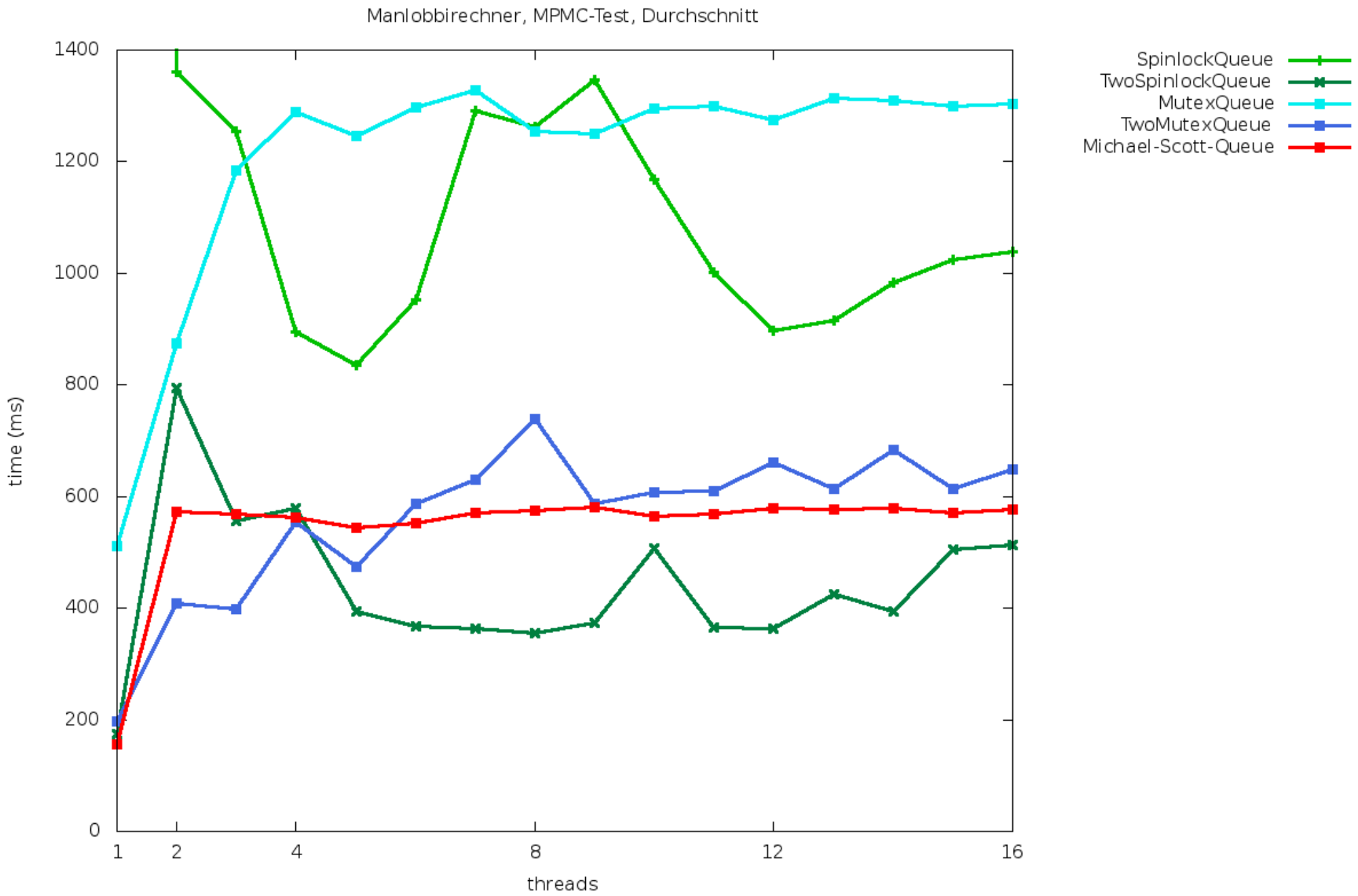
Ergebnisse


Bigbox, Roundtest(100% Elemente), Durchschnitt
time (ms)
threads
0
2000
4000
6000
8000
10000
12000
14000
1
4
8
12
24
36
48
60
72
84
96
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue

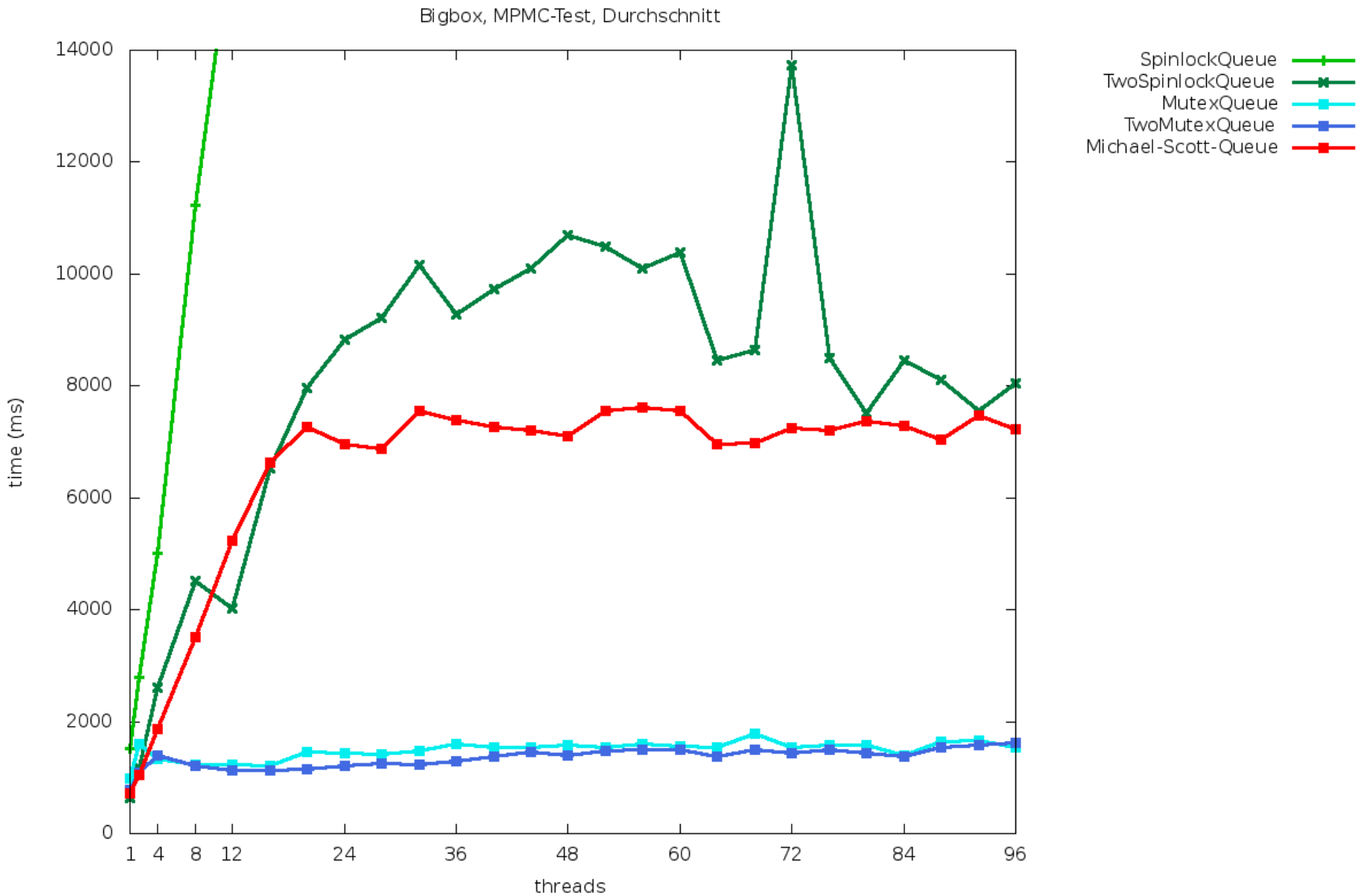
Ergebnisse


Haswellrechner, Roundtest(100% Elemente), Durchschnitt
time (ms)
threads
0
200
400
600
800
1000
1200
1400
1
2
4
8
12
16
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue
TransactionQueue
TwoTransactionsQueue

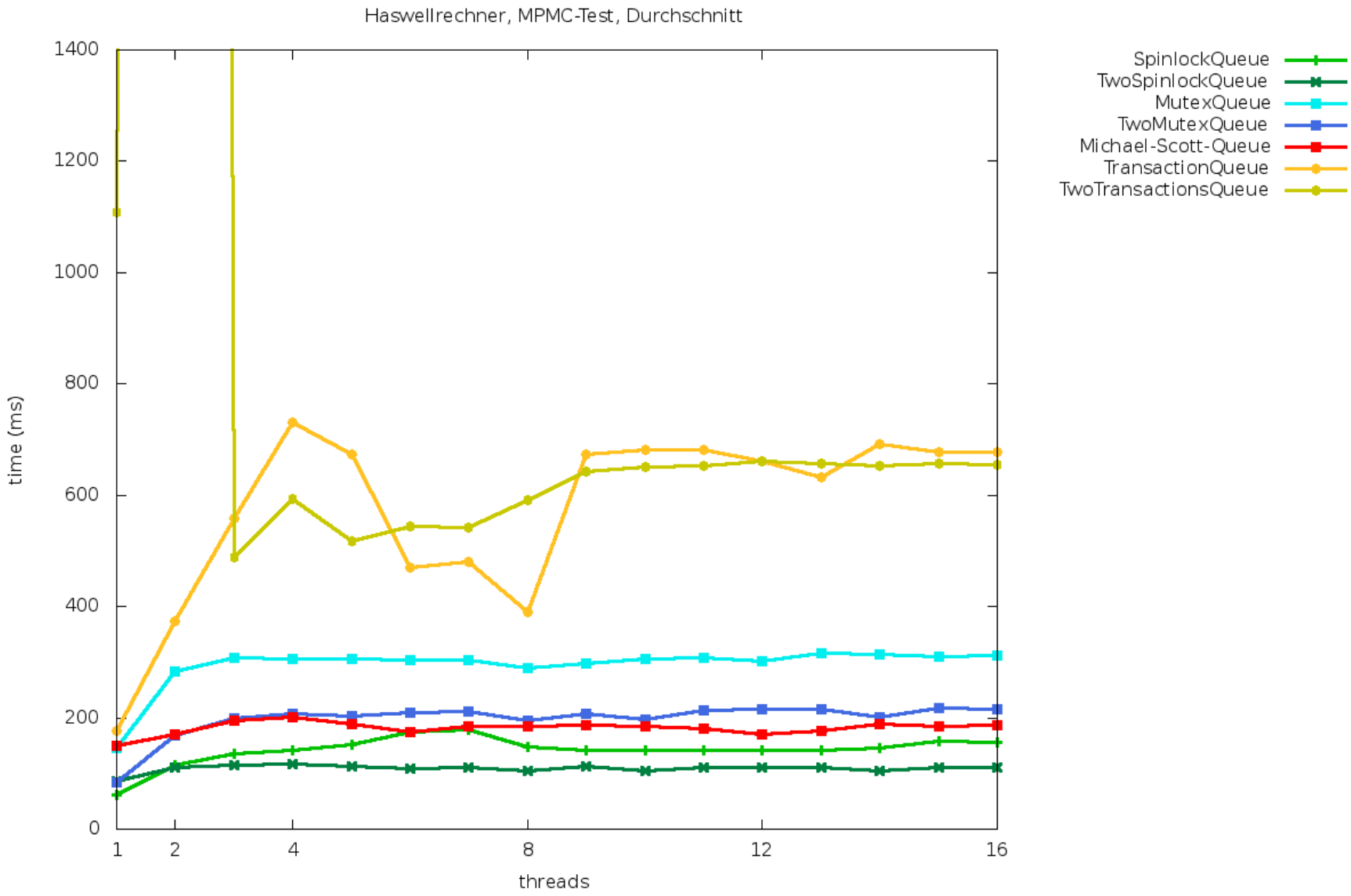
Ergebnisse


Manlobbirechner, MPMC-Test, Durchschnitt
time (ms)
threads
0
200
400
600
800
1000
1200
1400
1
2
4
8
12
16
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue

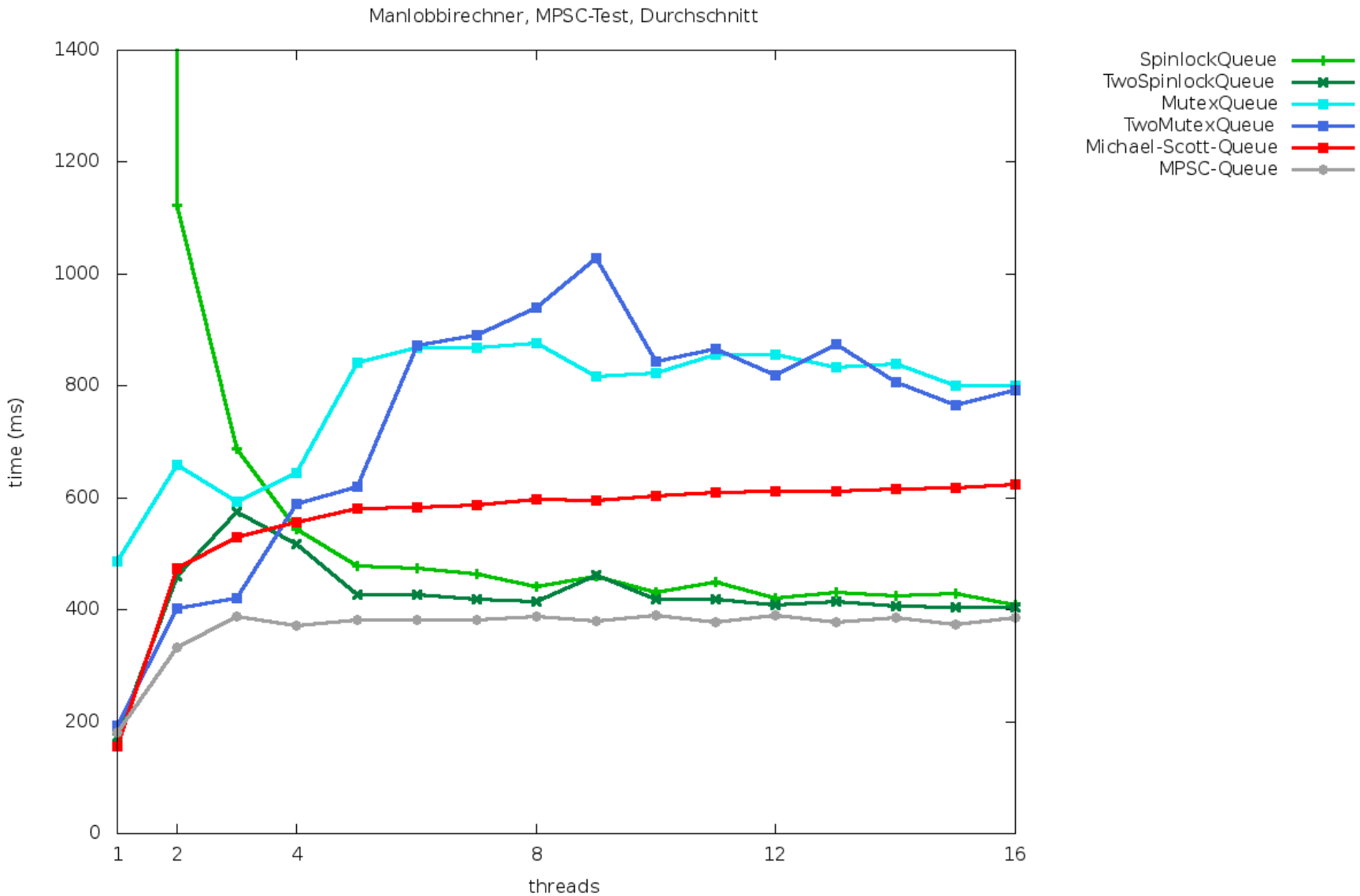
Ergebnisse


Bigbox, MPMC-Test, Durchschnitt
time (ms)
threads
0
2000
4000
6000
8000
10000
12000
14000
1
4
8
12
24
36
48
60
72
84
96
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue

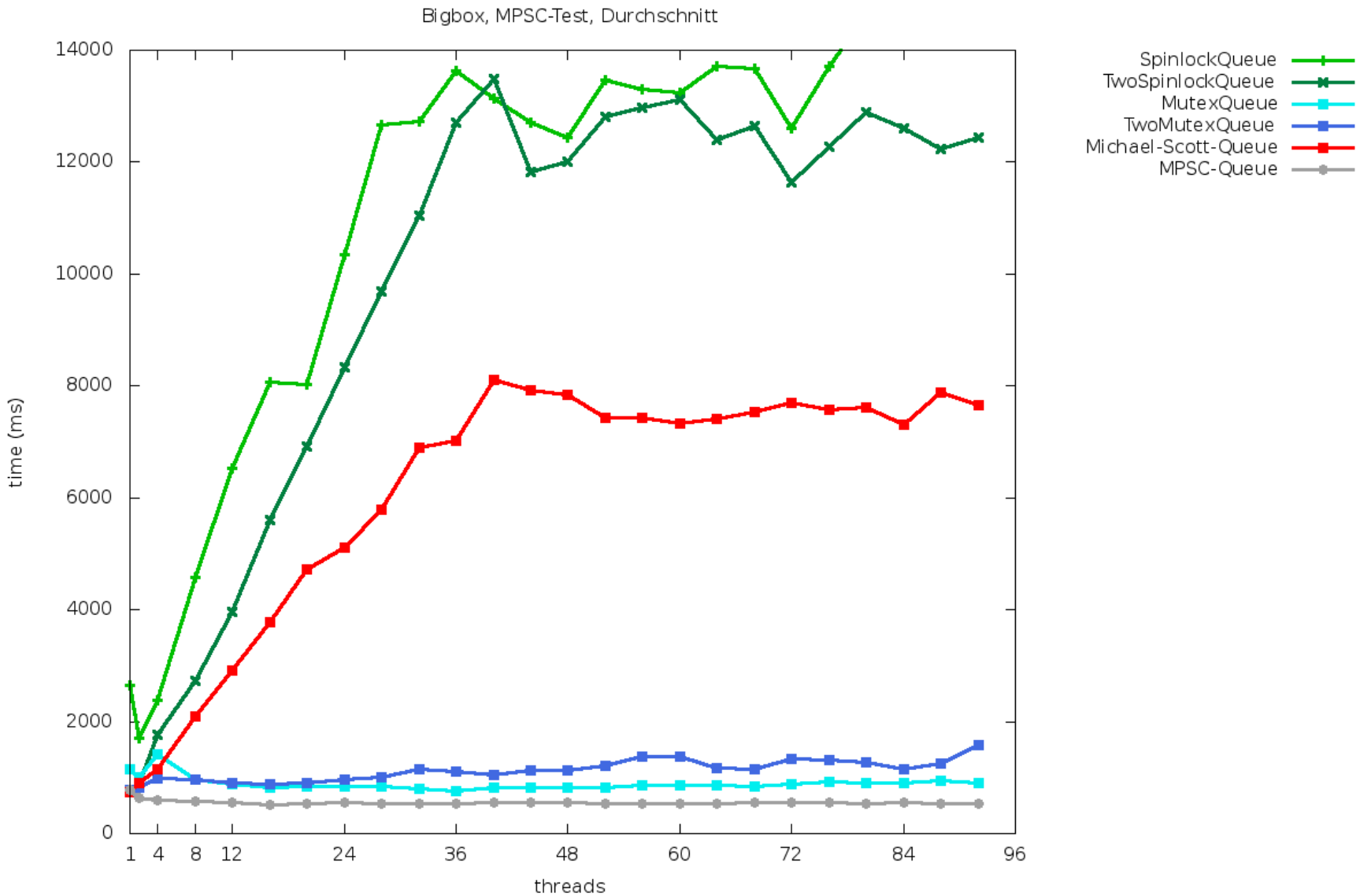
Ergebnisse


Haswellrechner, MPMC-Test, Durchschnitt
time (ms)
threads
0
200
400
600
800
1000
1200
1400
1
2
4
8
12
16
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue
TransactionQueue
TwoTransactionsQueue

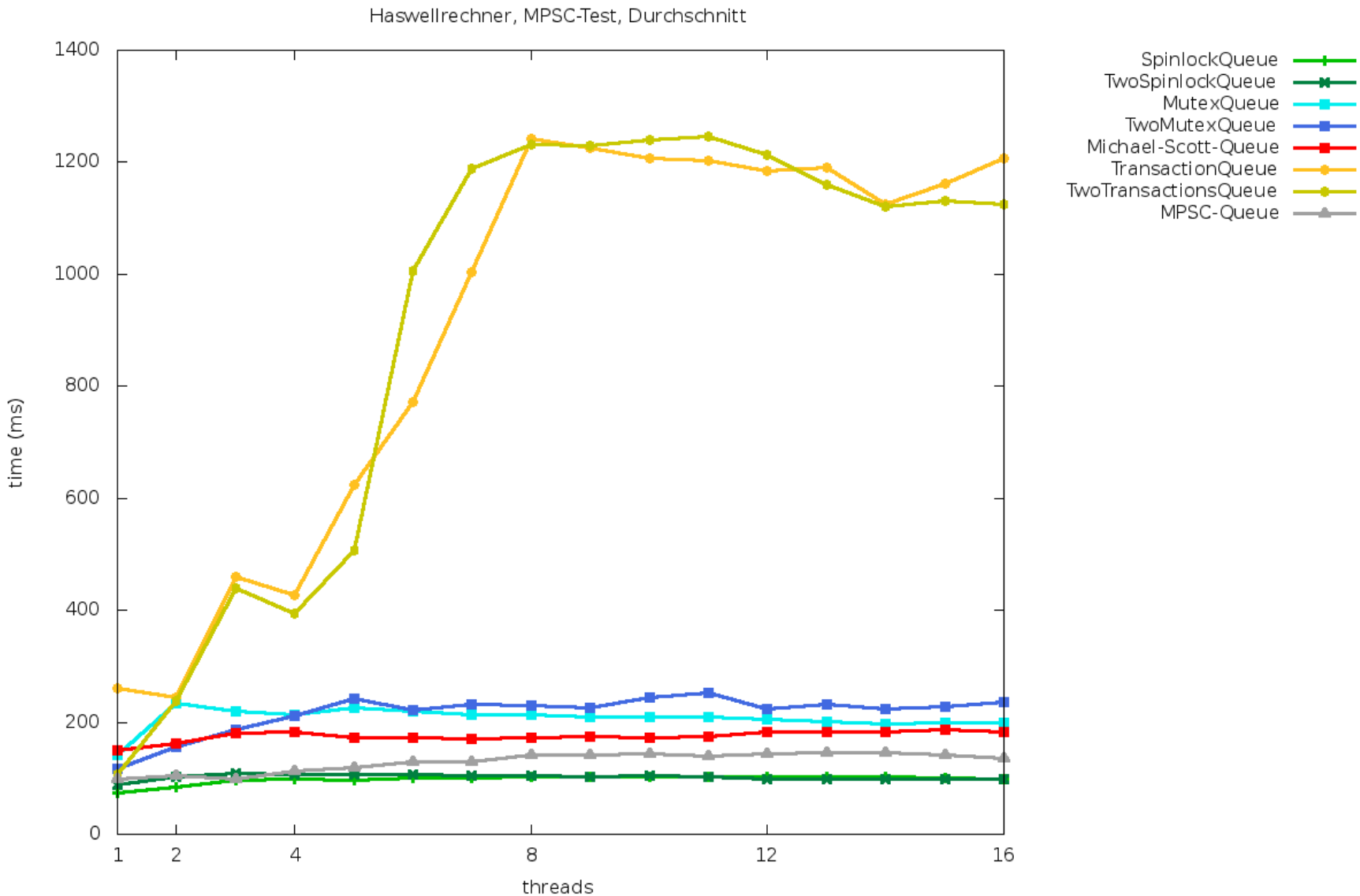
Ergebnisse


Manlobbirechner, MPSC-Test, Durchschnitt
time (ms)
threads
0
200
400
600
800
1000
1200
1400
1
2
4
8
12
16
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue
MPSC-Queue

Ergebnisse


Bigbox, MPSC-Test, Durchschnitt
time (ms)
threads
0
2000
4000
6000
8000
10000
12000
14000
1
4
8
12
24
36
48
60
72
84
96
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue
MPSC-Queue

Ergebnisse


Haswellrechner, MPSC-Test, Durchschnitt
time (ms)
threads
0
200
400
600
800
1000
1200
1400
1
2
4
8
12
16
SpinlockQueue
TwoSpinlockQueue
MutexQueue
TwoMutexQueue
Michael-Scott-Queue
TransactionQueue
TwoTransactionsQueue
MPSC-Queue

Fazit

- Michael-Scott Queue ist durchgehend suboptimal.
  Allerdings gibt es auch immer schlechtere Varianten.
  Insgesamt ist sie damit in jeder Hinsicht mittelmäßig.
- MPSC-Queue ist auf den meisten Systemen besser als blockierende Queues, aber auf Haswell Architektur wird sie von der Spinlock Queue überholt.
- Insgesamt ist es nicht offensichtlich, welche Queue unter welchen Bedingungen optimal arbeitet.
- Messdaten, Testumgebung und verwendete Queues auf:
  git://github.com/Lorenz-badgers/qtpie
  https://github.com/Lorenz-badgers/qtpie.git

Erfahrungen

- nachträgliche Implementierung gemeinsamer Schnittstelle zwischen verschiedenen Queues und Testsystem komplexer als erwartet
- Messungen für präsentierte Graphen später als geplant durchführbar
- Umgang mit Messdaten: mehr Daten speichern, um z.B. Streuungen ermitteln zu können

Vielen Dank für Ihre Aufmerksamkeit!