

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>15</b>
1.1	Für wen ist dieses Buch?	15
1.2	Ziele des Buchs	16
1.3	Parallel Computing: Warum?	17
1.3.1	CPU: Umkehrung der Entwicklung	18
1.3.2	Multi-core-Prozessoren	21
1.3.3	Nutzung von Multi-core-Prozessoren	21
1.4	Grundlagen	23
1.4.1	Definition: Parallel Computing	25
1.4.2	Rechnerarchitekturen	26
1.4.3	Multithreading vs. Parallel Computing	31
1.4.4	Asynchrone Programmierung vs. Parallel Computing	32
1.4.5	Arbeitsweise	33
1.4.6	Parallele Programmiermodelle	33
1.4.7	Einteilung nach Flynn	37
1.5	Performanceindikatoren und Gesetzmäßigkeiten	38
1.5.1	Speedup	39
1.5.2	Effizienz	41
1.5.3	Amdahlsches Gesetz	42
1.5.4	Gustafson-Gesetz	45
1.5.5	Mehraufwand (Parallel Overhead)	45
1.5.6	Kritische Bereiche (Load Imbalance)	47
1.5.7	Slowdown-Effekt	48
1.5.8	Weitere wichtige Begriffe	48

1.6	Granularität	50
1.6.1	Fine-grain Parallelism	51
1.6.2	Coarse-grain Parallelism	51
<b>2</b>	<b>Allgemeine Konzepte</b>	<b>53</b>
2.1	Regeln für erfolgreiches Parallel Computing	53
2.1.1	Arbeitsverteilung	55
2.1.2	Zustandsverwaltung (Shared State)	60
2.1.3	Selbstblockade (Deadlock)	64
2.1.4	Starvation	69
2.1.5	Fehlerbehandlung	69
2.2	Projektmanagement und Planung	71
2.2.1	Grad der Parallelisierung	71
2.2.2	Kostenkalkulation	73
2.2.3	Anforderungsdefinition	73
2.3	Modellierungsmöglichkeiten	74
2.3.1	(Passive) Klassen und aktive Klassen	75
2.3.2	Kommunikation	77
2.3.3	Synchronisierung	79
<b>3</b>	<b>Die Basis: Threads unter .NET</b>	<b>81</b>
3.1	Das Prozessmodell unter Windows	81
3.1.1	Anatomie eines Threads	84
3.1.2	Speicherzuordnung	86
3.1.3	Kontextwechsel und State Transition	89
3.2	Ein Thread-Objekt erstellen	90
3.2.1	Erstellung eines Threads (kernel32.dll)	91
3.2.2	Erstellung eines Threads unter .NET	96
3.2.3	Managed Threads vs. Windows Threads	98
3.2.4	Zustände eines Threads	101

3.3	Verwendung von .NET Threads	103
3.3.1	Warten auf einen Thread	103
3.3.2	Steuerung von Threads	105
3.3.3	Threads beenden	106
3.3.4	Thread-Parameter	109
3.3.5	Background Threads vs. Foreground Threads	113
3.4	Atomare Operationen	116
3.4.1	Die Methoden der Interlocked-Klasse	121
3.5	Speichermodelle	123
3.5.1	Das Schlüsselwort volatile	124
3.5.2	VolatileWrite und VolatileRead	126
3.6	Speicherzugriff und Verwaltung	127
<b>4</b>	<b>Synchronisation von Threads</b>	<b>131</b>
4.1	Monitor	134
4.1.1	Hinweise zur Verwendung von Monitor	135
4.1.2	Die Monitor-Klasse richtig verwenden	137
4.1.3	Erweiterte Techniken: Pulse und Wait	139
4.1.4	Das Schlüsselwort lock	141
4.2	Mutex	142
4.2.1	Zugriffsrechte	144
4.2.2	Mutex vs. Monitor	146
4.3	Semaphore	147
4.4	Ereignisse (Events)	150
4.4.1	ManualResetEvent	150
4.4.2	AutoResetEvent	153
4.4.3	EventWaitHandle	153
4.5	ReaderWriterLock	156

<b>5</b>	<b>Erweiterte Thread-Techniken</b>	<b>161</b>
5.1	Warten auf Threads	161
5.2	Parameter und Ergebnisse	163
5.3	Verwendung des Threadpools	165
5.3.1	Threadpool-Varianten	167
5.3.2	Verwendung des CLR-Threadpools	168
5.3.3	Arbeitsweise des CLR-Threadpools	171
5.3.4	Lastverteilung	172
5.4	Kontrollierter Thread-Abbruch	175
5.4.1	Abbruch mittels Interrupt	177
5.5	Fehlerbehandlung	179
5.6	Ein Chatserver	180
5.6.1	Architektur und Funktionsweise	182
5.6.2	Der Chatserver	184
5.7	Client/Server-Protokoll	188
5.7.1	Serververwaltung	189
5.7.2	Chatclient	191
5.7.3	Zusammenfassung	193
<b>6</b>	<b>Task Parallel Library</b>	<b>195</b>
6.1	Bestandteile	200
6.1.1	Data Structures	201
6.1.2	Concurrency Runtime	202
6.1.3	Tools (Werkzeuge)	205
6.1.4	Programmiermodell	205
6.2	Das Task-Konzept	206
6.2.1	Zustände eines Tasks	207
6.2.2	Möglichkeiten der Task-Erzeugung	209
6.2.3	Optimale Ressourcennutzung	215
6.2.4	Race Conditions, Deadlocks und Ressourcen	220

6.3	Das Cancellation Framework	221
6.3.1	Motivation	222
6.3.2	Ziele	226
6.3.3	Funktionsweise und Architektur	227
6.3.4	Bestandteile	229
6.3.5	Verwendung	232
6.4	Erweiterte Task-Konzepte	238
6.4.1	Warten auf einen oder mehrere Tasks	238
6.4.2	Task Workflows (Prozesskette)	241
6.4.3	Task mit Rückgabewerten	243
6.5	Fehlerbehandlung	245
6.5.1	Verschachtelte Tasks	248
6.6	Schleifen	253
6.6.1	Parallel.For und Parellel.ForEach	254
6.6.2	Lokale Schleifenvariablen	255
6.6.3	Schleifenabbruch	258
6.6.4	Grad der Parallelität	260
6.6.5	Die Parallel.Invoke	260
6.7	Concurrent Data Structures	265
6.7.1	ConcurrentQueue<T>	266
6.7.2	ConcurrentStack<T>	268
6.7.3	ConcurrentBag<T>	271
6.7.4	BlockingCollection<T>	272
6.7.5	ConcurrentDictionary<TKey, TValue>	275
6.8	Sperren (Locks)	277
6.8.1	Barrier	278
6.8.2	CountdownEvent	281
6.8.3	ManualResetEventSlim	283
6.8.4	SemaphoreSlim	284
6.8.5	SpinWait und SpinLock	285

6.9	Lazy Initialization Classes	288
6.9.1	System.Lazy<T>	289
6.9.2	System.Threading.ThreadLocal<T>	291
6.9.3	System.Threading.LazyInitializer	292
<b>7</b>	<b>PLINQ – Parallel LINQ</b>	<b>297</b>
7.1	Funktionsweise	298
7.1.1	Ausführungsmodell	299
7.1.2	Merge Options	301
7.1.3	Partitionierung der Daten	302
7.1.4	With-Optionen	306
7.1.5	Fehlerbehandlung	312
7.2	Verwendung von PLINQ	313
7.2.1	Definition von Abfragen mittels Enumerable	314
7.2.2	Abfragemethoden	315
7.2.3	Query Expressions	316
7.3	LINQ zu PLINQ: Was ist zu beachten?	317
7.3.1	PLINQ und gemeinsame Ressourcen	317
7.3.2	PLINQ und Fehlerbehandlung	321
7.3.3	PLINQ und Reihenfolge	321
7.3.4	Struktur vs. Klasse	323
7.3.5	LINQ zu PLINQ: Einige Regeln	325
7.4	Zusammenfassung	326
<b>8</b>	<b>Erweiterte Task-Parallel-Techniken</b>	<b>327</b>
8.1	Task Scheduler (Task-Manager)	327
8.1.1	API-Übersicht	331
8.1.2	Langläufige Aufgaben	333
8.1.3	Inline Tasks	335
8.1.4	Steuerungsmöglichkeiten	335
8.1.5	FromCurrentSynchronizationContext	337

8.2	Benutzerdefinierte Aufgabenplaner	345
8.2.1	Task mit Priorität	346
8.2.2	Verwendung eigener Aufgabenplaner	351
8.2.3	Weitere Aufgabenplaner	352
8.3	TPL Dataflow (TDF)	352
8.3.1	Entstehung der TDF-Bibliothek	353
8.3.2	Aufbau und Funktionen der TDF	355
8.3.3	Umsetzung der TDF	356
8.3.4	Verwendung der Dataflow Blocks	361
8.3.5	Laufzeiteigenschaften	366
<b>9</b>	<b>Parallel Computing in Visual Studio 2010</b>	<b>371</b>
9.1	Threadauswertung	371
9.2	Parallel Tasks	372
9.3	Parallel Stacks	375
9.4	Performanceanalyse	377
9.4.1	CPU-Nutzung	381
9.4.2	Threadansicht	382
9.4.3	Verteilung auf CPU-Kerne	384
9.5	Profiling ohne Visual Studio 2010	385
9.5.1	Profiling durchführen	386
9.5.2	Laufenden Prozess auswerten	387
9.5.3	Auswertung der Leistungsdaten	387
9.6	Zusammenfassung	388
<b>10</b>	<b>Concurrency and Coordination Runtime</b>	<b>389</b>
10.1	Einordnung und Funktionsweise	390
10.1.1	CCR vs. Task Parallel Library und PLINQ	391
10.1.2	(Geschäfts-)Prozesse im Mittelpunkt	392
10.1.3	Datenflüsse und Kanäle	394
10.1.4	Vermeidung von Threads	395

10.2 Kernbestandteile	395
10.2.1 Ports	396
10.2.2 Arbitrer	400
10.2.3 Dispatcher und DispatcherQueue	403
10.3 Abbildung von Prozessen	409
10.3.1 Bedingungen und Prozessflüsse	410
10.3.2 Auswahl	412
10.3.3 Einfache Zusammenführung (2 Ports)	414
10.3.4 Warten auf mehrere Ports	416
10.3.5 Warten auf n Daten	419
10.4 Zusammenfassung	421
<b>11 Programmiermodelle</b>	<b>423</b>
11.1 Axum	424
11.1.1 Gemeinsame Ressourcen	425
11.1.2 Sperren – Synchronisation statt parallel	426
11.1.3 Vermeidung von Seiteneffekten durch Isolation	428
11.1.4 Axum basiert auf dem Actor Model	428
11.1.5 Verwendung von Axum	429
11.1.6 Domäne Agent Channel Port	431
11.1.7 Agent	432
11.1.8 Channel und Port	433
11.1.9 Datenflusskonzept	436
11.1.10 Request-Reply-Muster	439
11.1.11 Datendefinition für einen Channel	441
11.1.12 Gemeinsame Ressourcen	443
11.1.13 Zusammenfassung	445

## Inhaltsverzeichnis

---

11.2 Software Transaction Memory	445
11.2.1 Probleme klassischer Sperrmechanismen	447
11.2.2 Deadlock	448
11.2.3 Funktionsweise	448
11.2.4 Transaktionen und Rollback-Mechanismen	449
11.2.5 Verwendung	452
11.2.6 Weitere Funktionen	455
11.3 Asynchrone Programmierung	456
11.3.1 Bisherige asynchrone Ansätze	457
11.3.2 Probleme bisheriger asynchroner Ansätze	459
11.3.3 Task-basierter asynchroner Ansatz (TAP)	460
11.4 Parallel Computing und JavaScript	465
11.4.1 JavaScript im Browser	465
11.4.2 Web Workers	466
11.4.3 Zusammenfassung	470
<b>12 Zusammenfassung</b>	<b>471</b>
<b>Stichwortverzeichnis</b>	<b>473</b>