15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Die Parallel Extensions für das .NET Framework

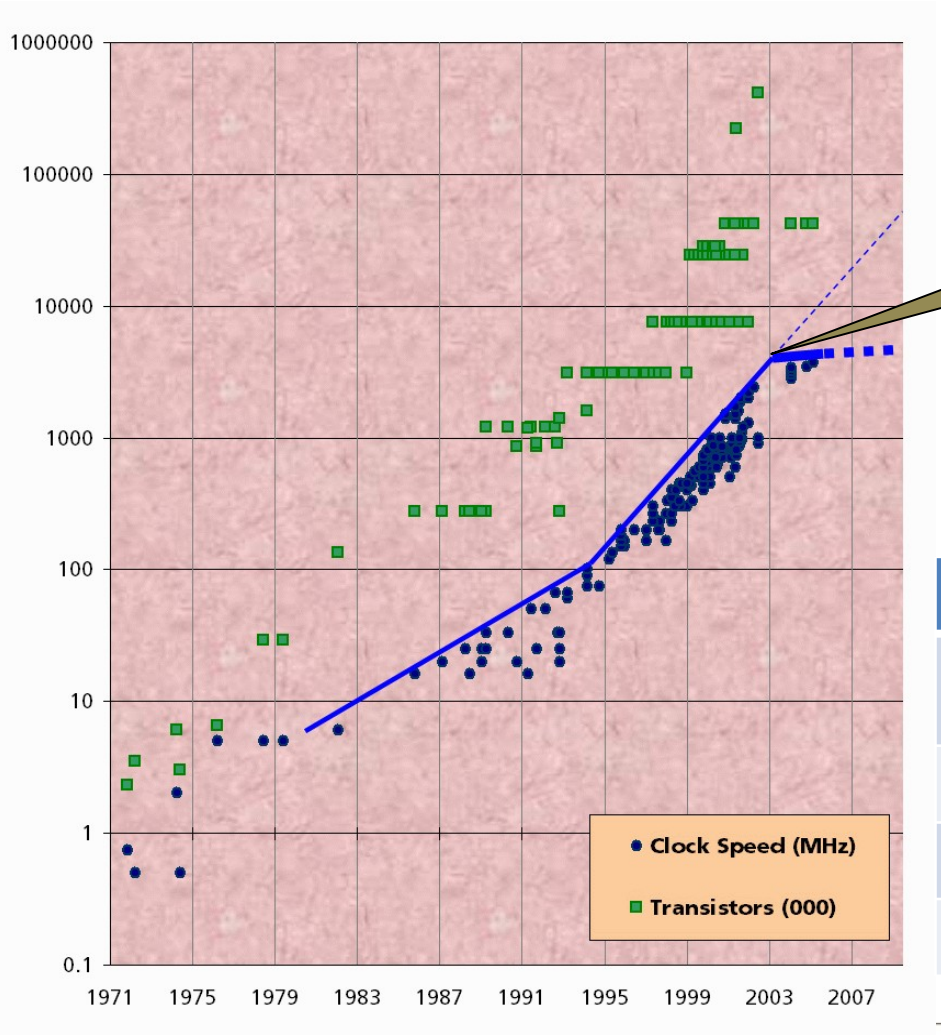
Softwareentwicklung für Multicore CPUs unter Microsoft .NET

Klaus Rohe,
klrohe@microsoft.com
Microsoft Deutschland GmbH

Agenda

- Einleitung
  - Free Lunch is over
  - Multicore- und Multiprozessor-Architekturen
- Multi-Threading
  - Probleme mit Multi-Threading
  - Parallelisierung & das Amdahlsche Gesetzt
- Microsoft Parallel Extensions to .NET
  - Task Parallel Library
  - PLINQ
- Ausblick & weitere Informationen

Taktrate von Intel CPUs

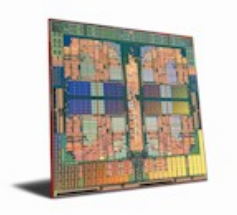


Only very slow linear increase CPU clock speed since 2003

Quelle: http://www.gotw.ca/publications/concurrency-ddj.htm

Intel Pentium 4 Prozessor	
Fläche	135 mm² = 1,35 cm² = 0,000135 m²
Leistungsaufnahme	115 W bei 3.6 GHz
Leistungsdichte (W / m²)	850 kW / m²
Entspricht einem schwarzen Strahler mit T = 1700 °C	

“The Free Lunch is Over”

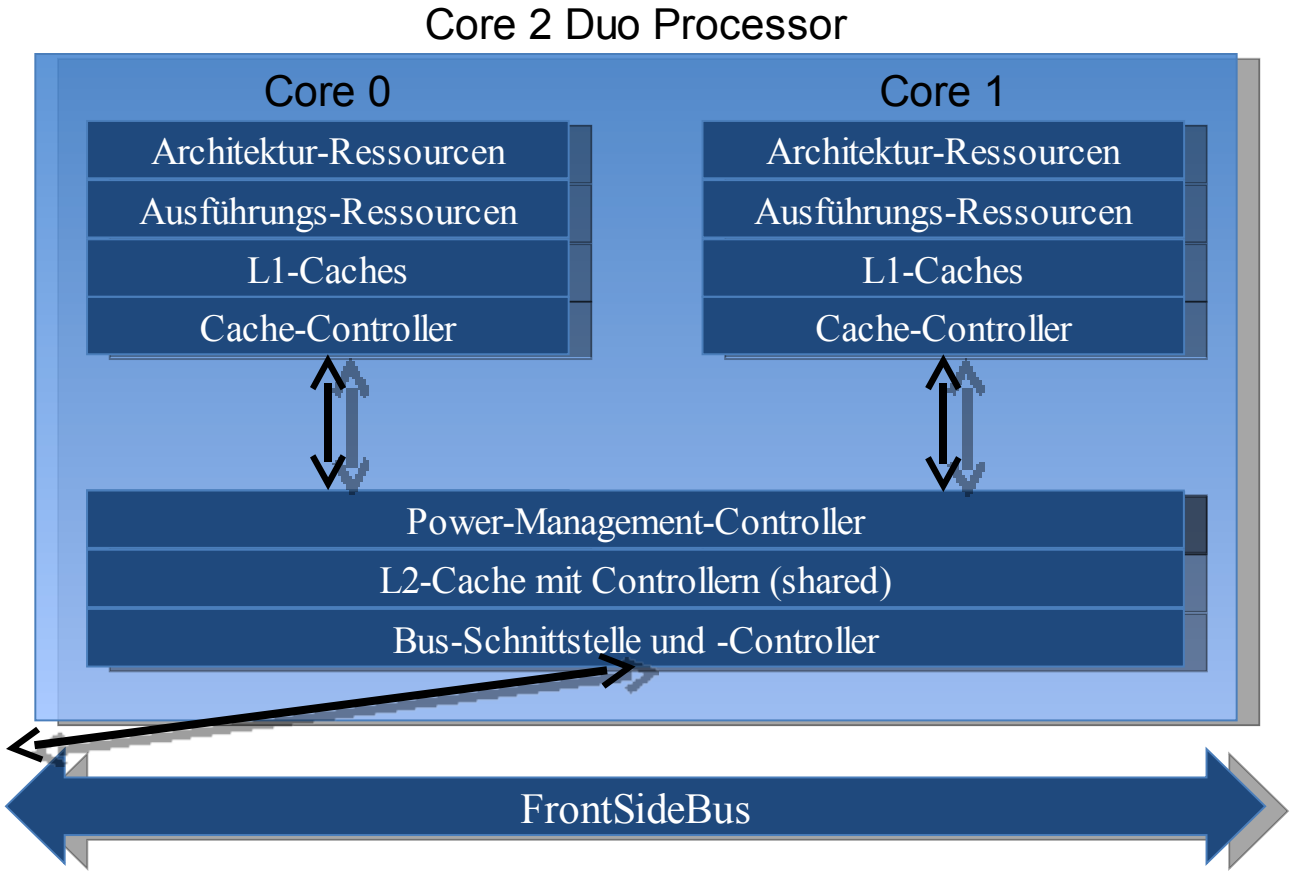


- Applikationen werden nicht performanter, wenn man die nächste CPU Generation nimmt

  The Free Lunch Is Over
  A Fundamental Turn Toward Concurrency in Software, By Herb Sutter
  http://www.gotw.ca/publications/concurrency-ddj.htm

- Normale Laptops und Desktops besitzen heute Multicore CPUs
- Single-Threaded Applikationen werden zu langsam
- Softwareentwickler müssen Multi-Threading nutzen!!
  - Aber…

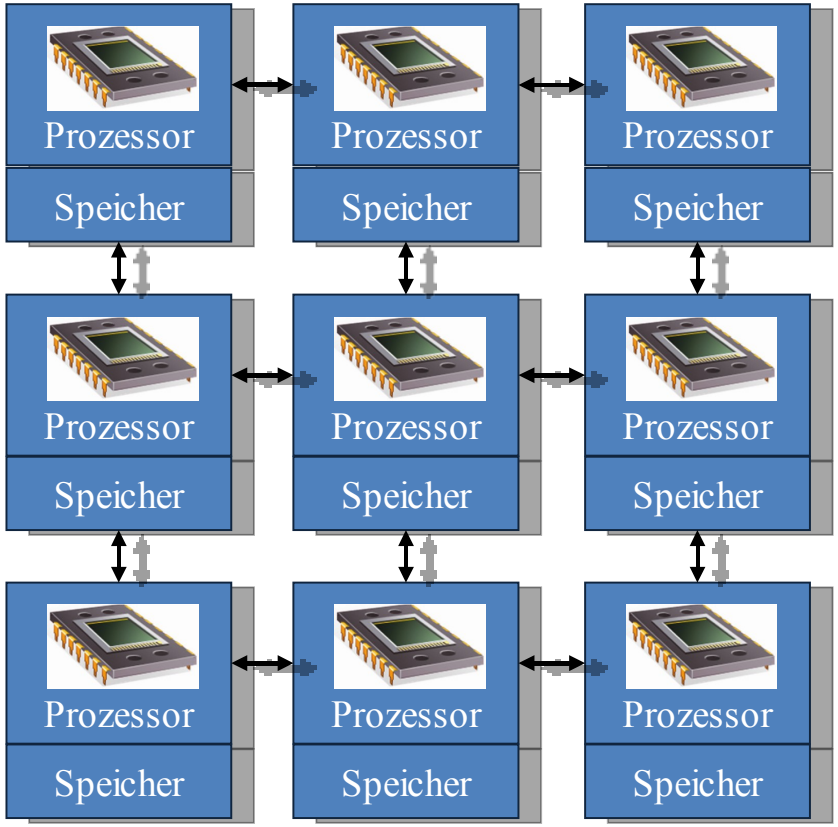
Architektur von Multicore-Prozessoren (Intel Core 2 Duo)



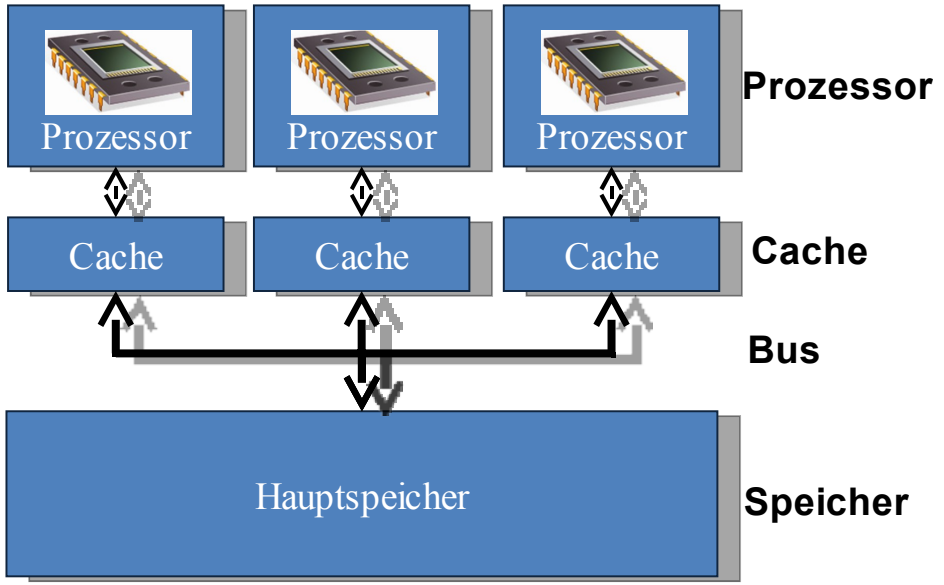
Aus: T. Rauber, G. Rünger, Multicore: Parallele Programmierung, Heidelberg 2008, ISBN 978-3-540-73113-9, Seite 15

Multiprozessor Architekturen

NUMA Architektur



SMP Architektur



- NUMA: **N**on **U**niform **M**emory **A**ccess
- SMP: **S**ymetrical **M**ulti **P**rocessor

Agenda

- Einleitung
  - Free Lunch is over
  - Multicore- und Multiprozessor-Architekturen
- Multi-Threading
  - Probleme mit Multi-Threading
  - Parallelisierung & das Amdahlsche Gesetzt
- Microsoft Parallel Extensions to .NET
  - Task Parallel Library
  - PLINQ
- Ausblick & weitere Informationen

Multi-Threading

- Mehrere Ausführungsstränge in einem Programm.
- Unterstützung durch die meisten, modernen Programmiersprachen
  - C/C++ Pthreads
  - Java
  - C#
  - Python
  - …
- Konstrukte zum:
  - Erzeugen von Threads
  - Synchronisieren von Threads
  - Sperren (Locken) von Ressourcen, die von mehreren Threads gemeinsam genutzt werden
  - Verwaltung von Threadpools

.NET Multi-Threading (C#)

```
String message = "Hello world from thread " + i;
ThreadStart hello = delegate() { Console.WriteLine(message); };
Thread t = new Thread(hello);
t.Start();
```

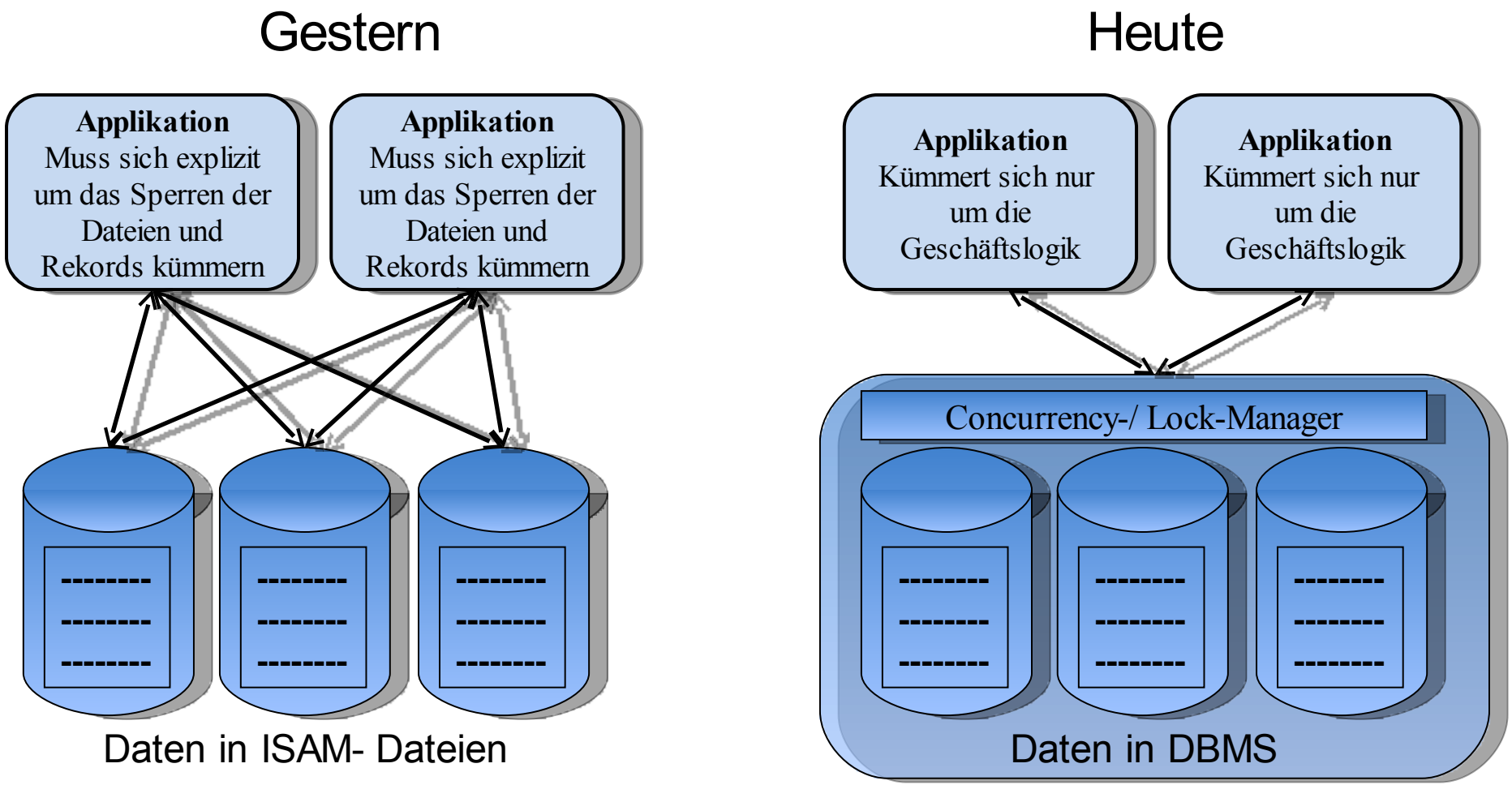
```
class Counter
 {
        int val;
        int GetAndIncrement()
        {
              lock (this)
              {
                    return val++;
              }
         }
}
```

```
public void Enq(T x)
{
     Monitor.Enter(this);
     try
     {
          while (tail - head == items.Length)
          {
               Console.WriteLine("===> Queue is empyty, waiting");
               Monitor.Wait(this); // queue is empty
          }
          items[(tail++) % items.Length] = x;
          Monitor.PulseAll(this);    // notify waiting dequeuers
          Console.WriteLine("===> Queue: {0} enqueued.", x);
     }
     finally
     {
          Monitor.Exit(this);
     }
}
```

Probleme mit Multi-Threading

- Die Programmierung von Anwendungen mit den üblichen Thread-Konstrukten wird sehr schnell, komplex und fehleranfällig werden.
  - Deadlocks, Race-Conditions, …
  - Performanceprobleme durch zu extensives Sperren von gemiensam genutzten Resourcen
- Fehler sind sehr schwer zu lokalisieren
  - Oft abhängig von der Last und Anzahl der CPUs
- Programmcode ist häufig schwer verständlich
- Resultat:
  - Schwer wartbare und teurere Software

Analogie: Datenbankprogrammierung gestern und heute



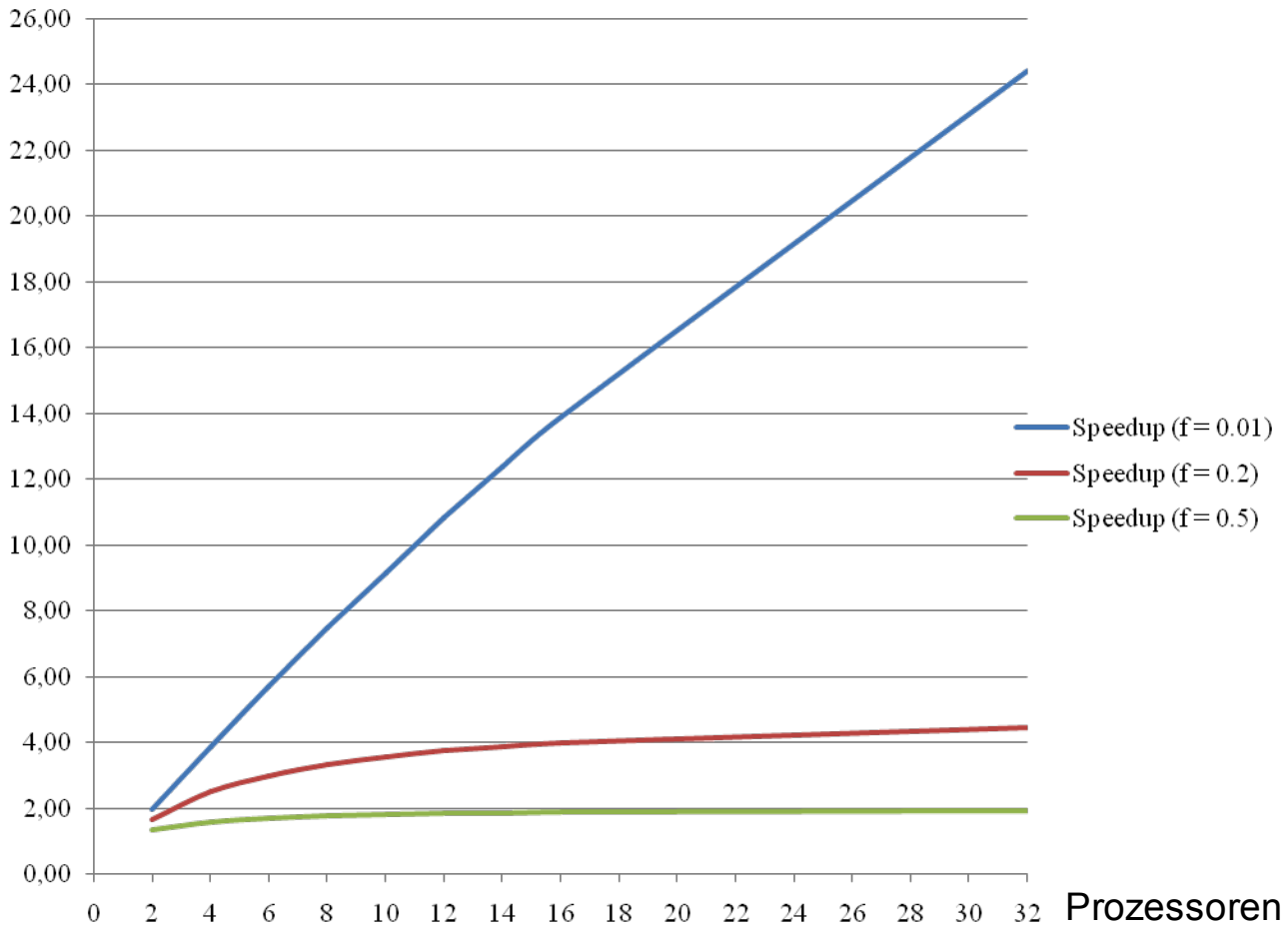
Parallelisierung & das Amdahlsche Gesetzt

- Die Anzahl der zur Verfügung stehenden Prozessoren sei p. Bei der parallelen Implementierung eines Programms, bei der ein Bruchteil f der Rechenzeit (0 <= f <= 1) sequentiell ausgeführt wird, beträgt der maximal erreichbare Speedup S:

$$S = 1 / (f + (1 - f) / p)$$

- Für f = 0.2 ergibt sich Beispielsweise : $S = 1 / (0.2 + 0.8 / p)$
  - Für p gegen Unendlich strebt S gegen 5. D. h. falls 20 % des Programmcodes sequentiell ist, ist der maximal erreichbare Speedup durch Parallelisieren 5! (Allgemein 1 / f)

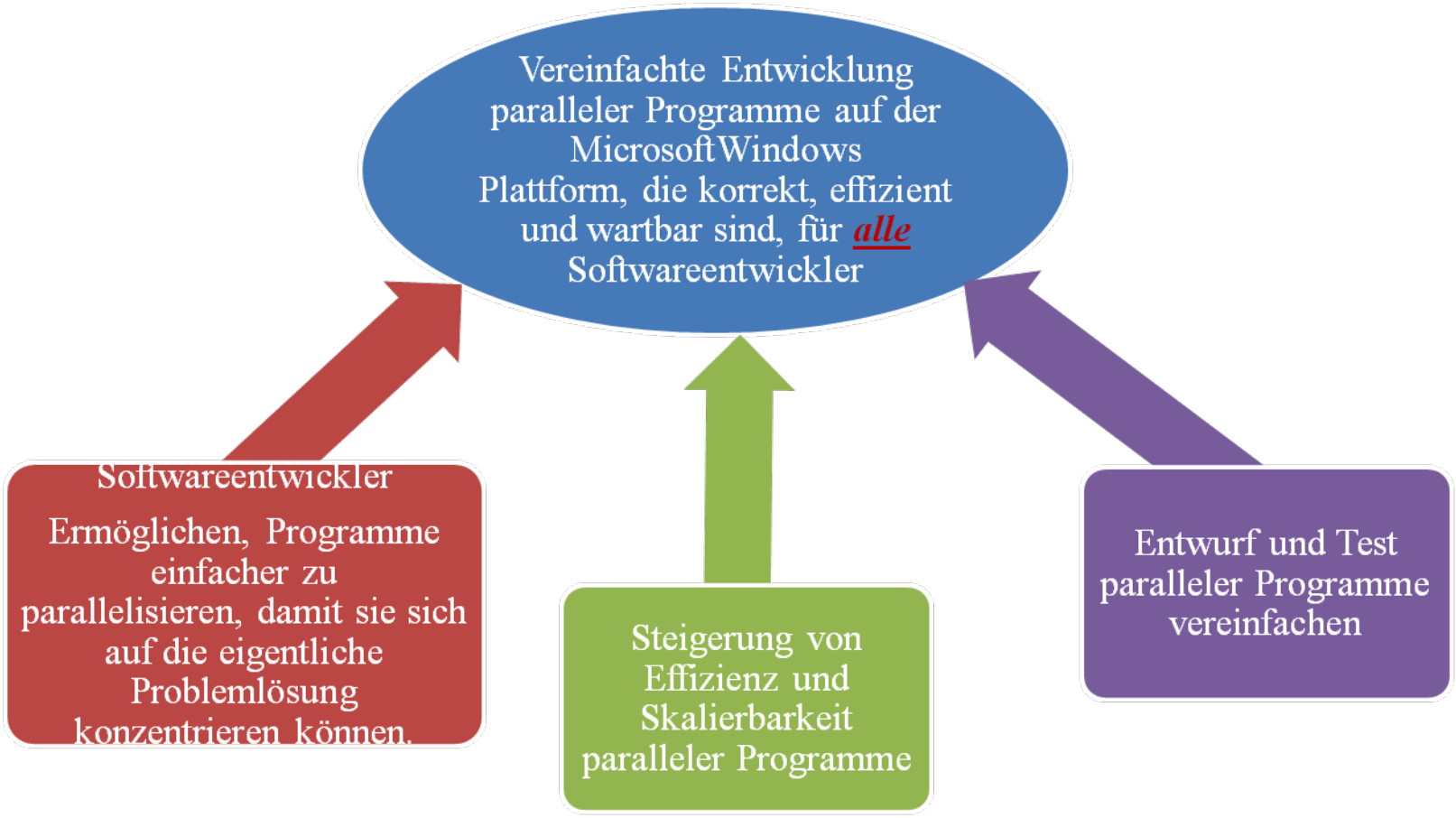
Das Amdahlsche Gesetzt (2)



Agenda

- Einleitung
  - Free Lunch is over
  - Multicore- und Multiprozessor-Architekturen
- Multi-Threading
  - Probleme mit Multi-Threading
  - Parallelisierung & das Amdahlsche Gesetzt
- Microsoft Parallel Extensions to .NET
  - Task Parallel Library
  - PLINQ
- Ausblick & weitere Informationen

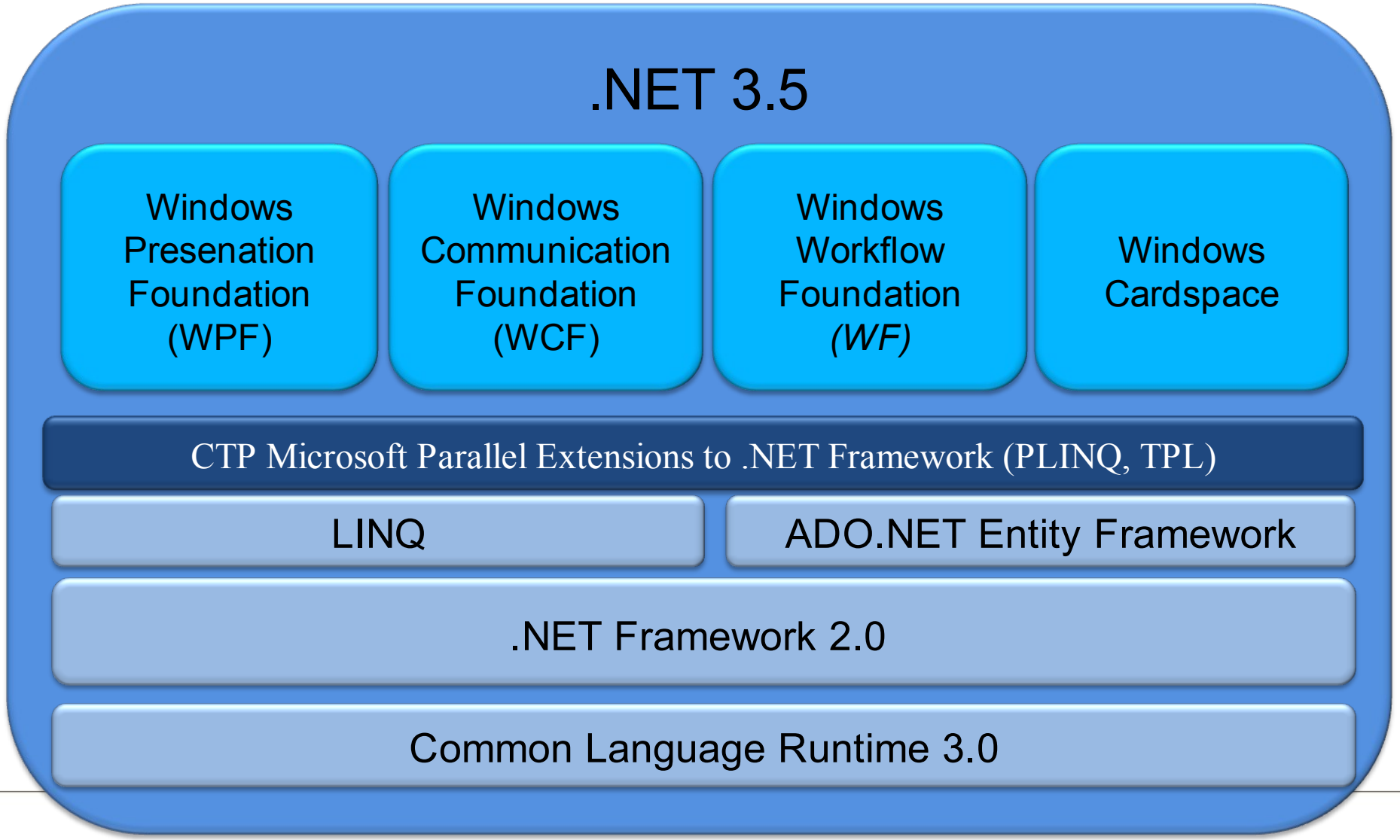
Microsoft Parallel Computing Initiative



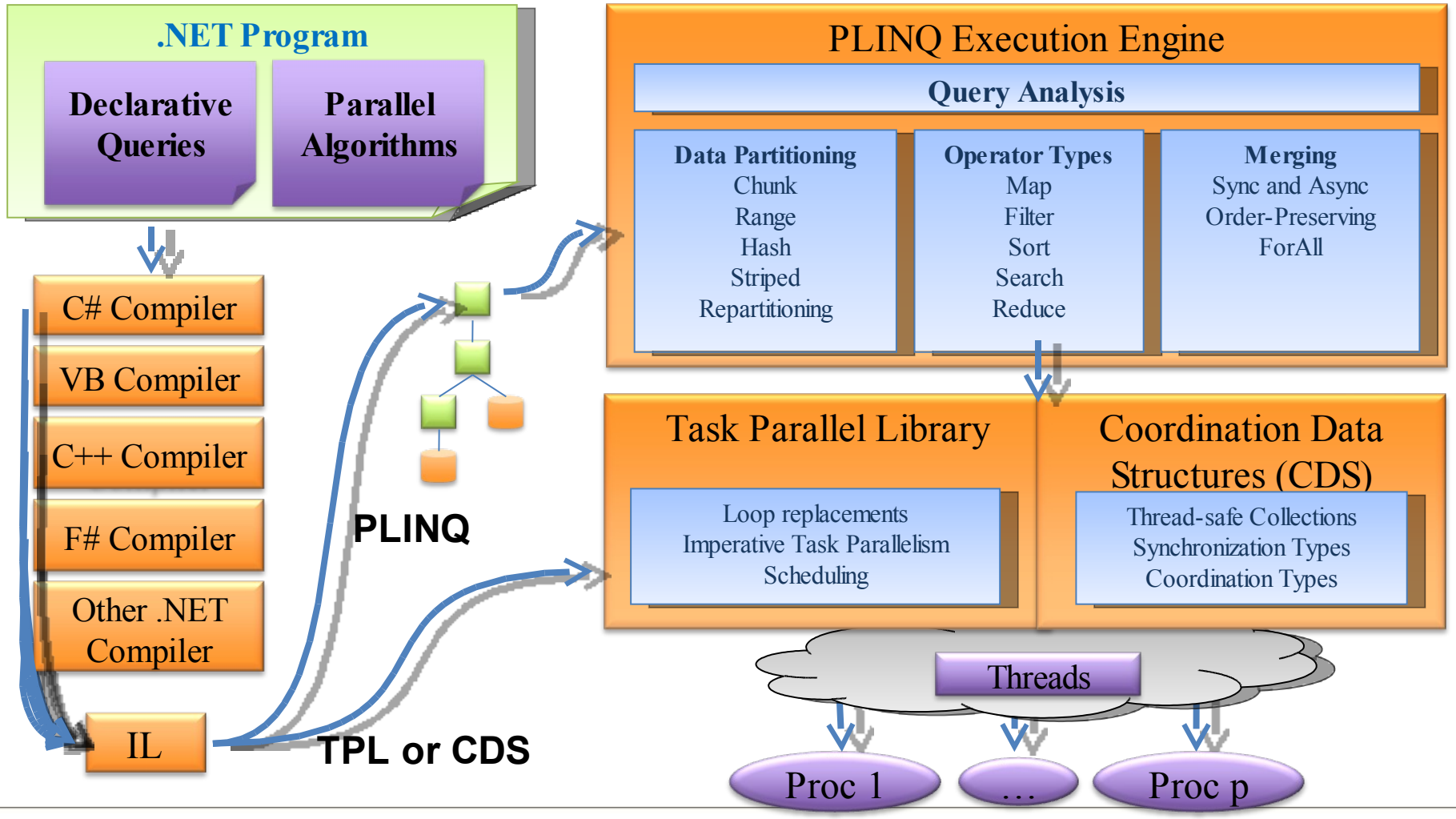
Parallel Extensions To The .NET Framework

- .NET Bibiliothek: keine Compiler Änderungen nötig
- Unterstützt deklaratives und imperatives Parallelisieren von Daten und Tasks:
  - Declarative data parallelism (PLINQ)
  - Imperative data and task parallelism (Task Parallel Library)
  - Coordination/synchronization constructs (Coordination Data Structures)
  - Gemeinsames Exception-Handling Modell

Microsoft Parallel Extensions to .NET Framework



Struktur der Parallel Extensions

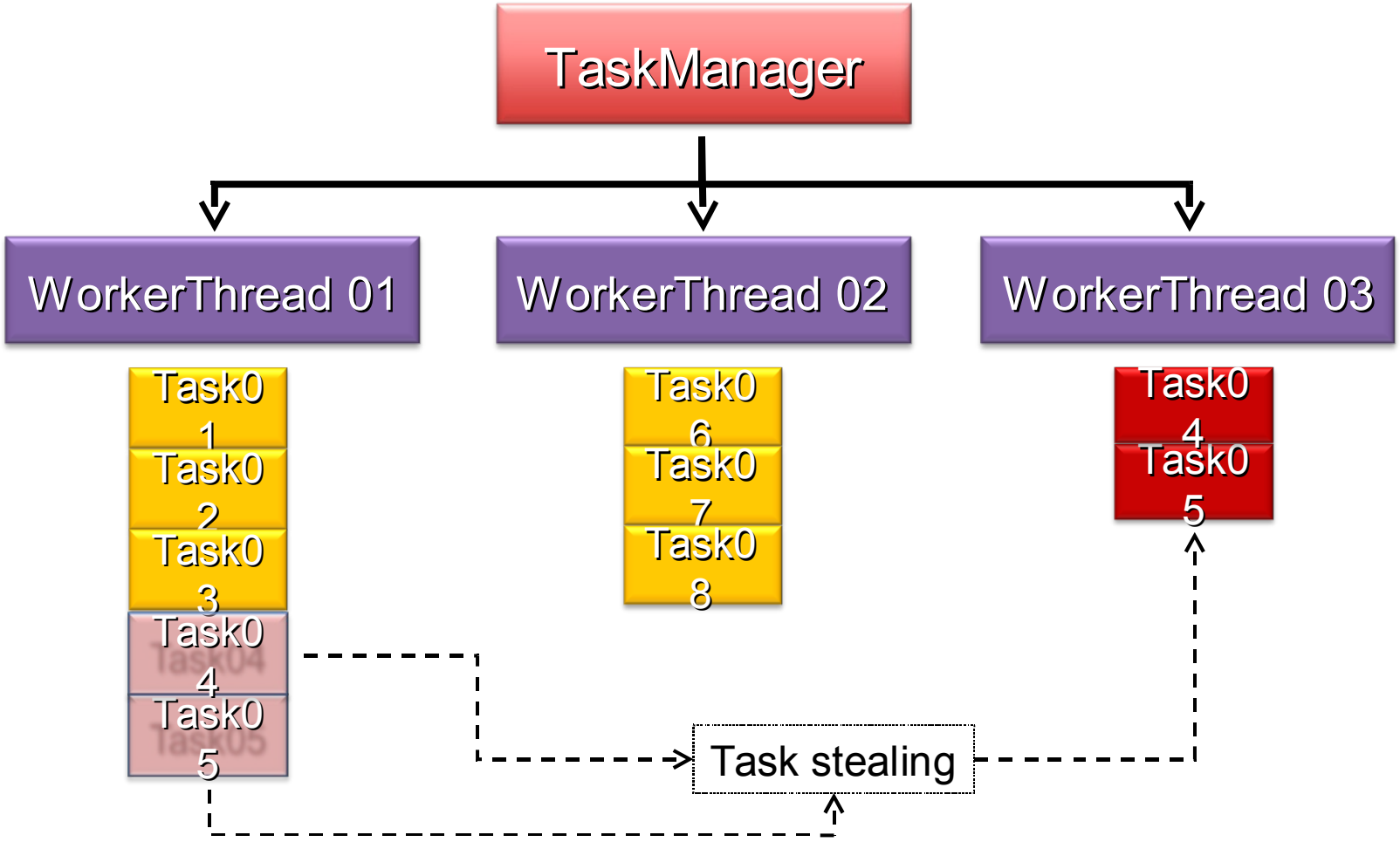


Tasks

- Ein zentrales Konzept der Parallel Extensions (PFX) ist die Task:
  - Anonyme Funktion, die unabhängig ausgeführt werden kann.
  - System.Threading.Tasks
  - Führt eine Funktion in einem separaten Thread aus.

```
class Program
{
        static void Main(string[] args)
        {
                Console.WriteLine("\nThread id = {0} in Main.“
                                Thread.CurrentThread.ManagedThreadId);

                Task task = Task.Create(delegate { PrintPrimes(10); });
                task.Wait();

                Console.Write("\nPress any key to exit!");
                Console.ReadKey();
        }

        private static void PrintPrimes(ulong n)
        {
                Console.WriteLine("\nThread id = {1} in PrintPrimes({0}).\n",
                                n, Thread.CurrentThread.ManagedThreadId);

                for (ulong i = 0; i <= n; i++)
                {
                        if (PrimeFunctions.IsPrime(i))
                                Console.WriteLine("{0} is prime", i);
                }
        }
}
```

TaskManager



Futures

- Ein „Future“ ist eine „Task“, mit einem Rückgabewerte

```
class Program
{
     static void Main(string[] args)
     {
          Random rnd = new Random();

          var ball1 = Future.Create(() => {  Thread.Sleep(rnd.Next(1000, 10000));
                                             return rnd.Next(1, 43); } );

          Console.WriteLine("Lottery in progress...");
          Console.WriteLine(ball1.Value);
          Console.ReadKey();
     }
}
```

Parallele Ausführung von Statements

- Folge von Programmstatements:

```
StatementA();
StatementB;
StatementC();
```

- Falls Programmstatements unabhängig sind, parallelisierbar:

```
Parallel.Invoke(
   () => StatementA(),
   () => StatementB,
   () => StatementC());
```

Parallele Schleifen: Parallel.For

```
// Sequentiell

void SeqMatrixMult(int size, double[,] m1, double[,] m2, double[,] result)
{
  for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
      result[i, j] = 0;
      for (int k = 0; k < size; k++) {
        result[i, j] += m1[i, k] * m2[k, j];
      }
    }
  }
}
```

```
// Mit TPL

using System.Threading;

void ParMatrixMult(int size, double[,] m1, double[,] m2, double[,] result)
{
  Parallel.For( 0, size, delegate(int i) {
    for (int j = 0; j < size; j++) {
      result[i, j] = 0;
      for (int k = 0; k < size; k++) {
        result[i, j] += m1[i, k] * m2[k, j];
      }
    }
  });
}
```

LINQ Beispiel

```
using System;
using System.Query;
using System.Collections.Generic;

class app
{
    static void Main()
    {
        string[] names = { "Burke", "Connor",
                "Frank", "Everett", "Albert", "George",
                "Harris", "David"};

        IEnumerable<string> expr = from s in names
                                   where s.Length == 5
                                   orderby s
                                   select s.ToUpper();

        foreach (string item in expr)
        {
            Console.WriteLine(item);
        }
    }
}
```

Parallel LINQ (PLINQ)

- LINQ: Language Integrated Query
- Ermöglicht LINQ Entwicklern parallele Hardware (Multicore-CPUs, …) auszunutzen
  - Unterstützt alle LINQ Query-Operatoren
- PLINQ abstrahiert von den Details der Parallelisierung
  - Partitioniert die Daten und führt sie automatisch wieder zusammen. („Klassische Datenparallelität“)
- Funktioniert für alle IEnumerable<T>

Coordination Data Structures (CDS)

- Concurrent Collections
  - Implementieren das Interface IConcurrentCollection<T>
    - ConcurrentQueue<T>
    - ConcurrentStack<T>
- BlockingCollections<T>
  - Unterstützt blockierendes Hinzufügen und Entfernen von Elementen falls Kapazitätsgrenzen unter- oder überschritten werden.
  - Kann IConcurrentCollection<T> als Speicherstruktur verwenden
- SpinWait
  - „Busy waiting“ ist machmal kostengünstiger als auf Synchronisation durch den Kernel zu warten.
- SpinLock
  - Basiert auf „Spinning“, kein Warten auf ein Kernel-Ereignis. Begründung s. o.
- SemaphoreSlim
  - Leichtgewichte Semaphor Implementierung

Demo

- Mandelbrot-Menge: Parallelisierung von Schleifen
- PLINQ: Filtern eines Arrays von Type Person.

```
public class Person
{
     public string FirstName { get; set; }

     public string LastName { get; set; }

     public string City { get; set; }

     public string State { get; set; }

     public DateTime BirthDate { get; set; }

     public double Age
     {
          get
          {
               TimeSpan tmsp = (DateTime.Now -
BirthDate);
               return tmsp.TotalDays;
          }
     }

     public bool HeavyTest()
     {
          ComputeintsiveMethod(1000);
          return true;
     }
     ...
}
```

Agenda

- Einleitung
  - Free Lunch is over
  - Multicore- und Multiprozessor-Architekturen
- Multi-Threading
  - Probleme mit Multi-Threading
  - Parallelisierung & das Amdahlsche Gesetzt
- Microsoft Parallel Extensions to .NET
  - Task Parallel Library
  - PLINQ
- Ausblick & weitere Informationen

Ausblick

- Neue Werkzeuge zum Entwerfen, Debuggen und Testen von parallelen Programmen
- Software Transactional Memory (STM) für die Synchronisation beim Zugriff auf gemeinsame Ressourcen
  - Siehe auch: SXM, C# Software Transactional Memory (Microsoft Research)
    - http://research.microsoft.com/research/downloads/Details/6cfc842d-1c1
- Weiterentwicklung der Concurrency Runtime (Vortrag von Ralf Westpahl auf dem Herbstcampus)
- Microsoft Professional Developer Conference 2008 (PDC) in Los Angeles vom 26.10. bis zum 27.10.2008
  - http://www.microsoftpdc.com/

Weitere Informationen (1)

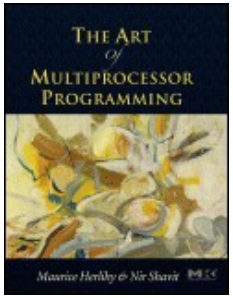
- Parallel Computing on msdn:
  - http://msdn.microsoft.com/en-us/concurrency/default.aspx
- Parallel FX June 2008 CTP Download:
  - http://www.microsoft.com/downloads/details.aspx?FamilyId=348F73FD-593D-4B3C-B055-694C50D2B0F3&displaylang=en#filelist
- Optimize Managed Code For Multi-Core Machines
  - http://msdn.microsoft.com/en-us/magazine/cc163340.aspx
- PLINQ:
  - http://msdn.microsoft.com/en-us/magazine/cc163329.aspx#S1

Weitere Informationen (2)

- Parallel FX team blog:
  - http://blogs.msdn.com/pfxteam/
- Multicore Crisis?
  - http://blog.mischel.com/2008/08/04/multicore-crisis/
- T.Rauber, G.Rünger, Multicore: Parallele Programmierung, Heidelberg 2008, ISBN 978-3-540-73113-9
- S. Gochman et al, Introduction to Intel Core Duo Processor Architecture, Intel Technology Journal, 10(2), Seite 89 – 97, 2006

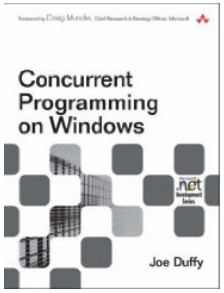
Weitere Informationen (3)

- M. Herlihy, Nir Shavit, The Art of Multiprocessor Programming, Morgan Kaufmann, ISBN-13: 978-0-12-370591-4, 528 Seiten



- T. Rauber, G. Rünger, Multicore: Parallele Programmierung, Springer, ISBN: 978-3-540-73113-9, 164 Seiten

- Joe Duffy, Concurrent Programming on Windows: Architecture, Principles, and Patterns, Addison Wesley, ISBN-13: 978-0321434821, 1008 Seiten
  - (Angekündigt für Nov. 2008)



.NET und Visual Studio - weitere Angebote

MSDN – Das Microsoft Developer Network

- Nachrichten & Informationen für Entwickler rund um Microsoft Technologien
- www.msdn-online.de

Xtopia

- 17.-18. November 2008 in Berlin
- Die Microsoft-Konferenz für Business, Web Technology, Design & UX.
- Programm und Anmeldung unter: www.xtopia-konferenz.de

Microsoft Technical Summit 2008

- 19.-21. November 2008 in Berlin
- Microsoft-Technologien, -Produkte und -Services von heute und morgen!
- Programm und Anmeldung unter: www.technical-summit.de

Visual Studio Team System Information Day

- Regelmäßige ganztägige Informationsveranstaltung von Microsoft
- Praxisnahe Demos & viel Raum für Diskussionen
- Details & Anmeldung: www.event-team.com/events/visualstudio

Vielen Dank!

Klaus Rohe,
klrohe@microsoft.com
Microsoft Deutschland GmbH

LINQ Beispiel

```
using System;
using System.Query;
using System.Collections.Generic;

class app
{
    static void Main()
    {
        string[] names = { "Burke", "Connor",
                "Frank", "Everett", "Albert", "George",
                "Harris", "David"};

        IEnumerable<string> expr = from s in names
                                   where s.Length == 5
                                   orderby s
                                   select s.ToUpper();

        foreach (string item in expr)
        {
            Console.WriteLine(item);
        }
    }
}
```