Parallel Computing

Christopher Schleiden

5.6.2009

Microsoft Student Partners

Agenda

- Einführung & Motivation
- Nativer Code
  - Threads
  - OpenMP
- .NET
  - Threads
  - ThreadPool
  - Parallel Extensions / TPL
  - PLINQ

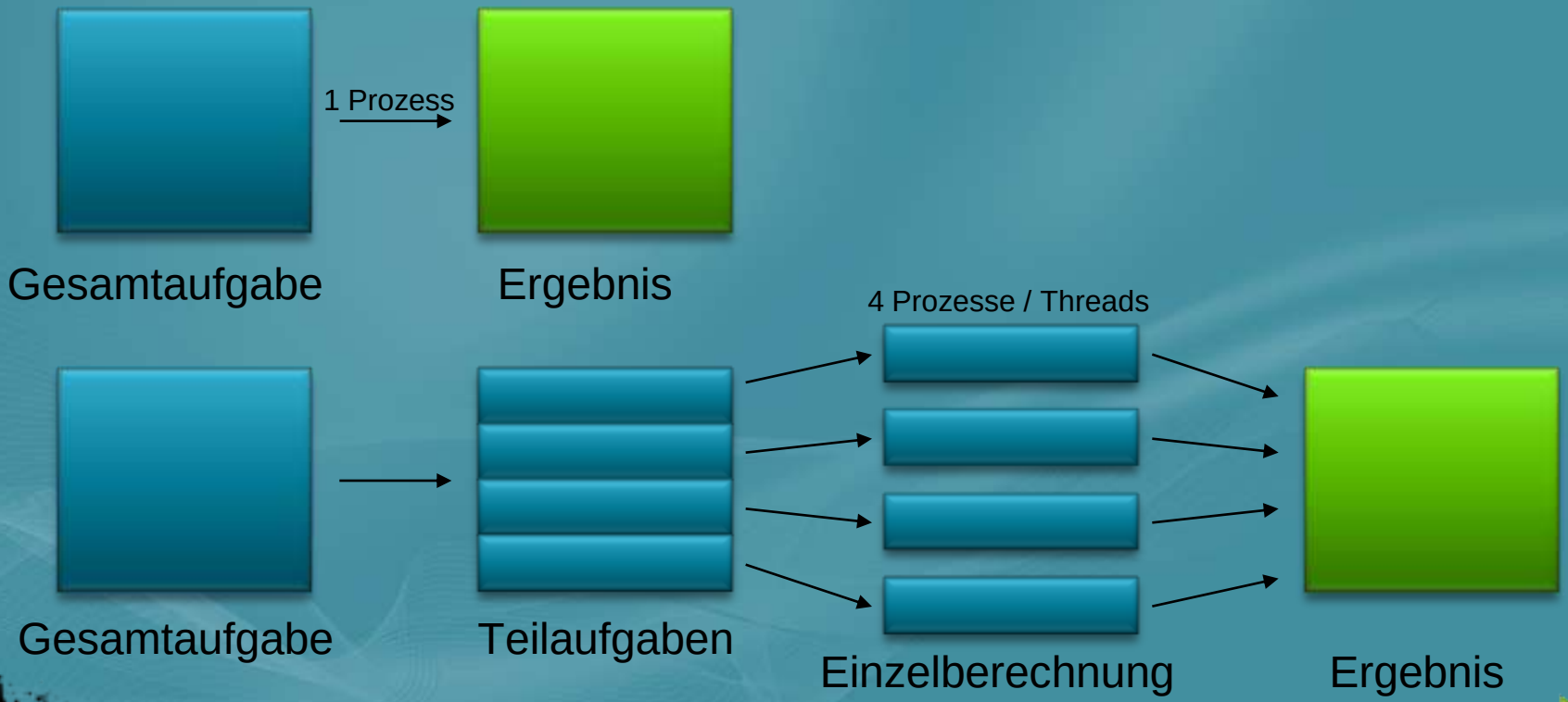
EINFÜHRUNG & MOTIVATION

Microsoft Student Partners

Was ist Parallel Computing?

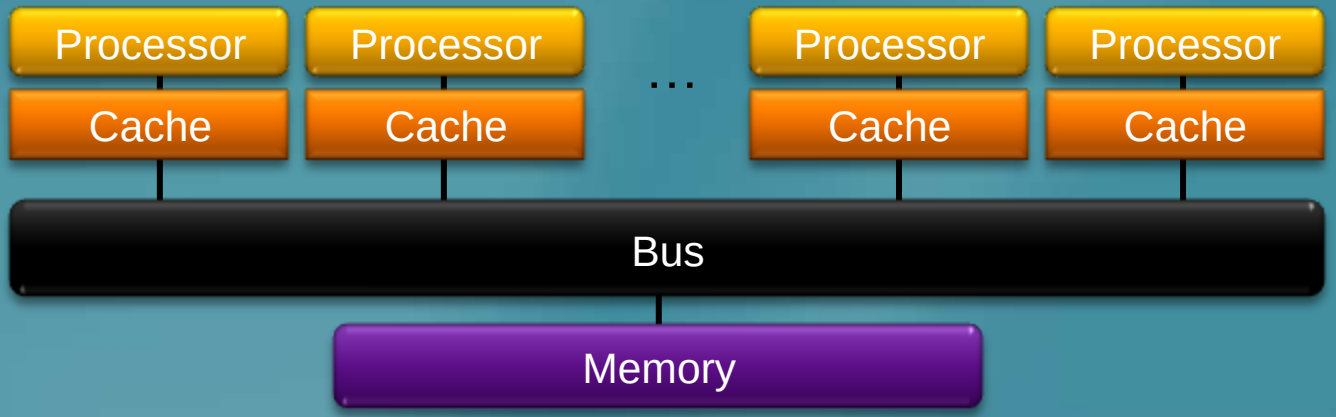
- Ganz allgemein:

  Parallele Ausführung von Code

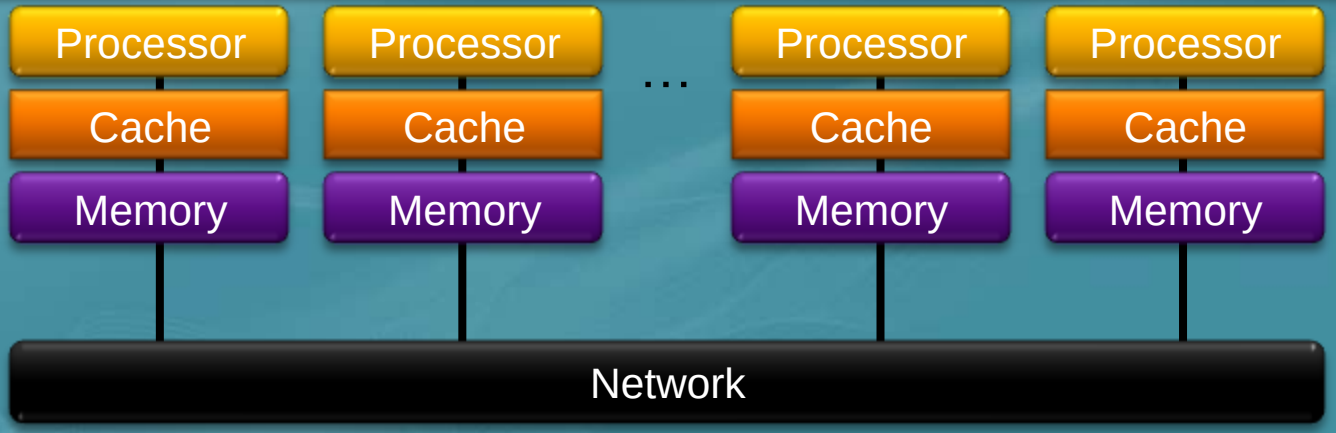


Shared vs. Distributed Memory

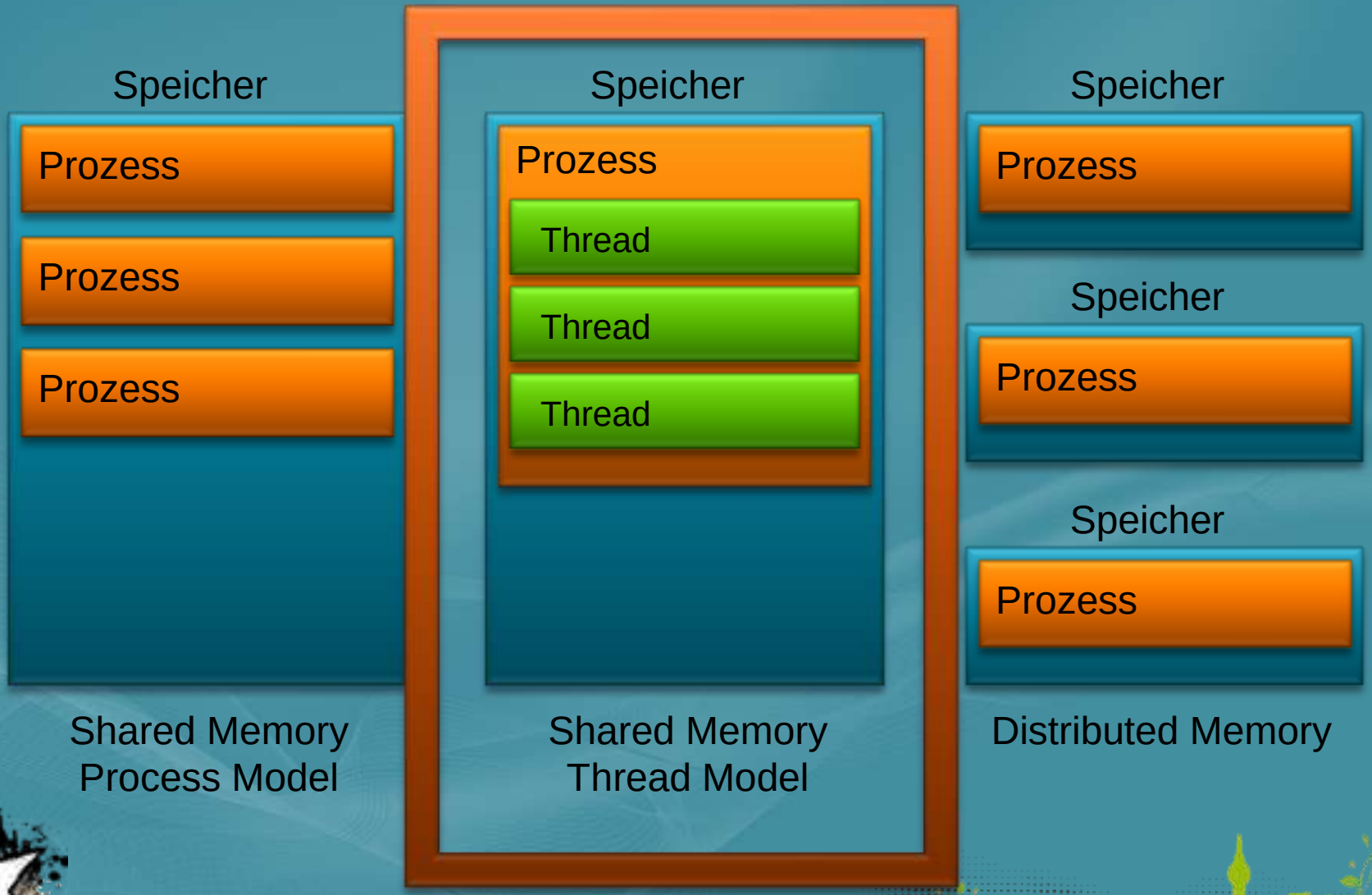
- Shared



- Distributed

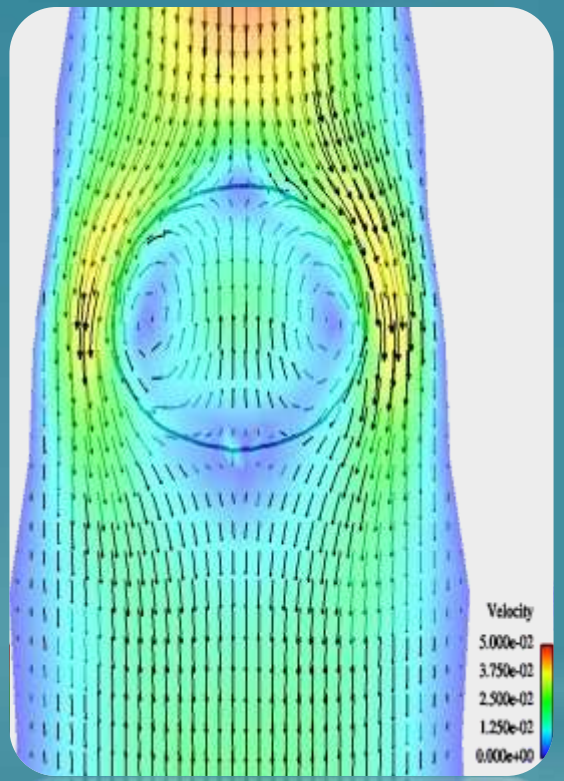


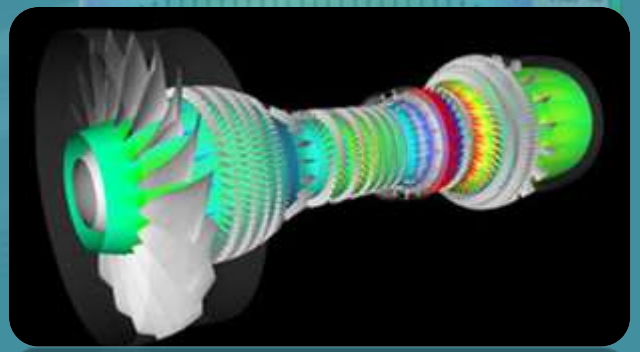
Shared vs. Distributed Memory



Einsatzgebiete

- Verschiedene Einsatzgebiete
  - Klassisch:
    High-Performance Computing
    - Wettersimulation
    - Finite Elemente Methode
    - Computational Fluid Dynamics
  - Aber auch:
    Benutzerinterfaces/Usability
    - Latenz vermeiden
    - *usw…*





Warum Parallel Computing?

Warum soll(t)en sich alle Entwickler mit Parallel Computing beschäftigen?

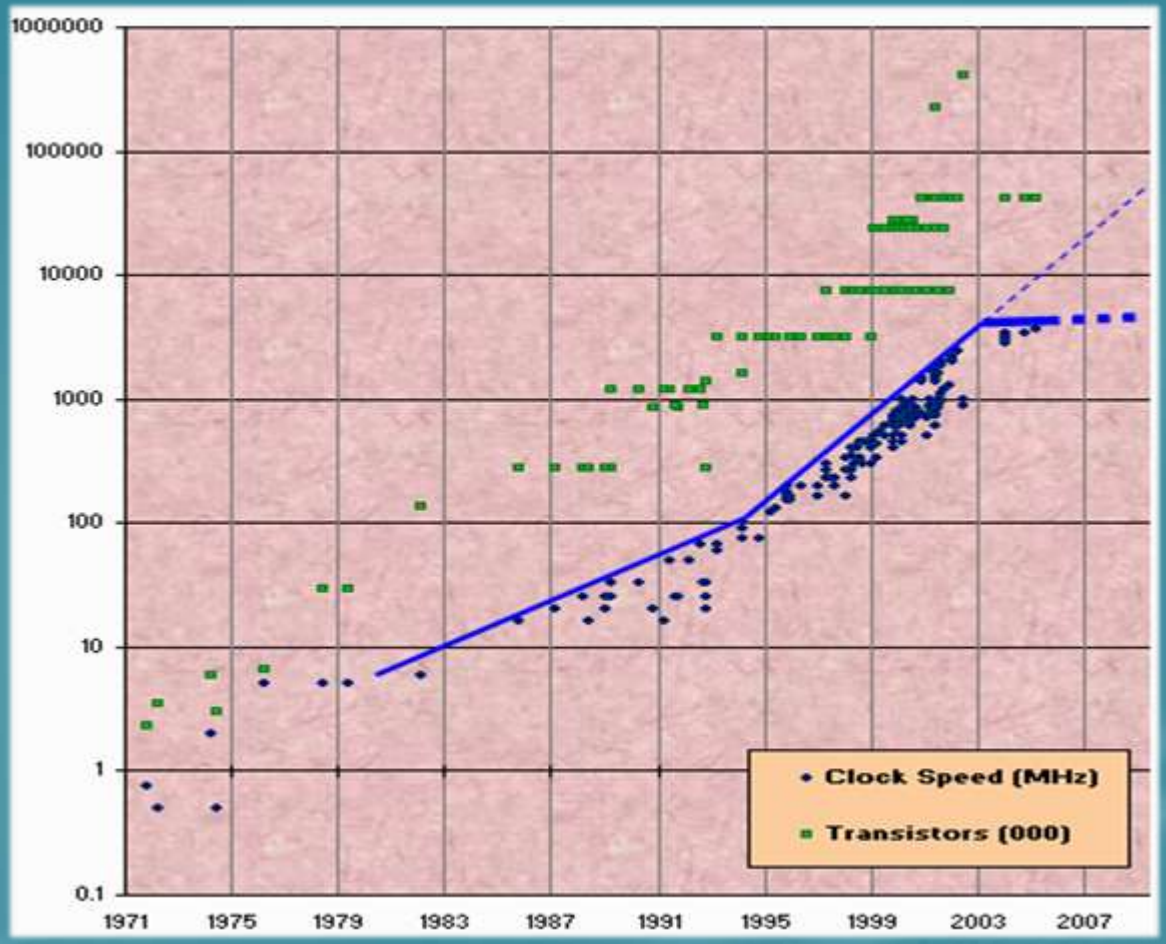
Warum Parallel Computing?

Warum soll(t)en sich alle Entwickler mit Parallel Computing beschäftigen?

Warum Parallel Computing?

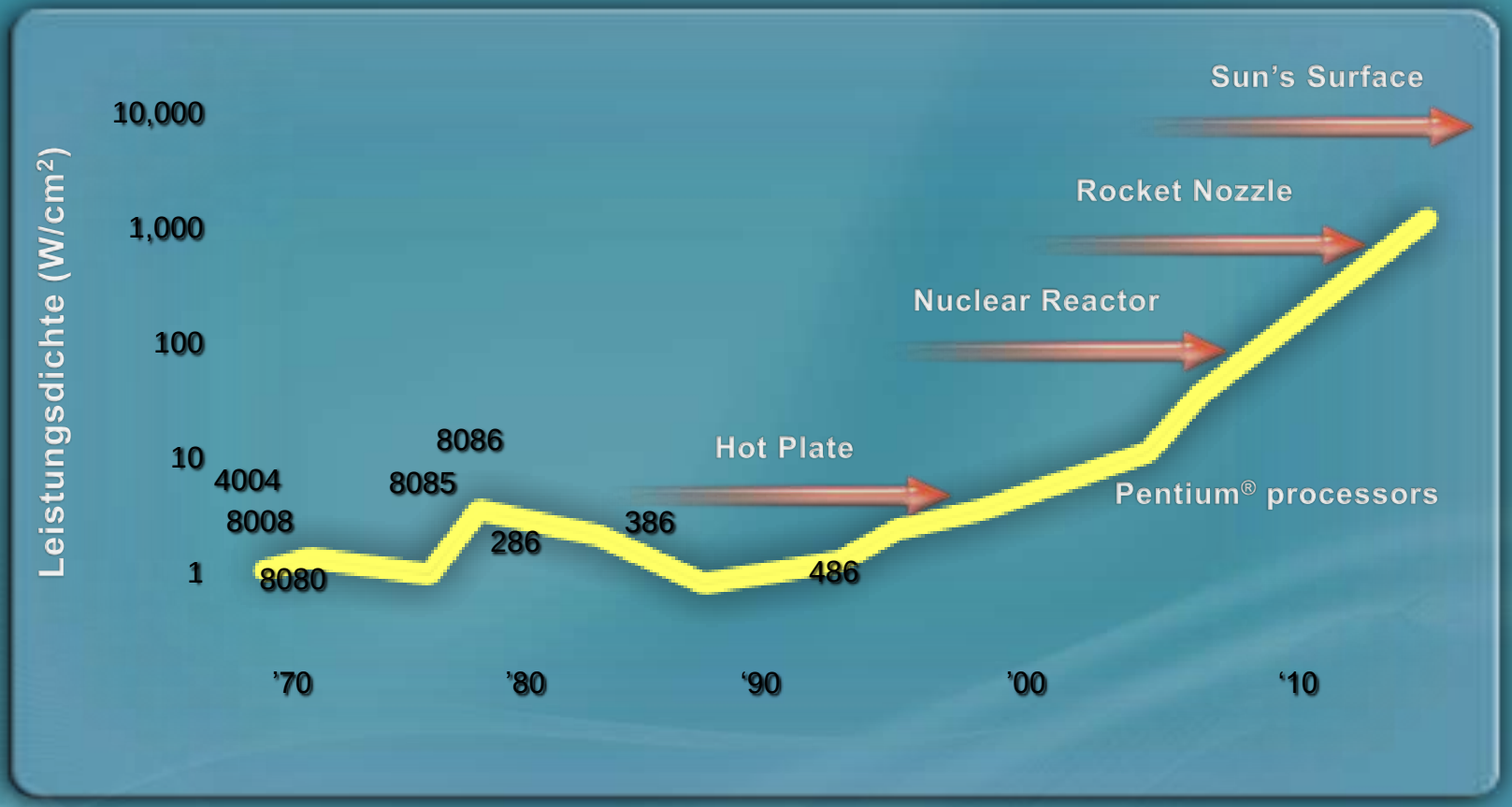
„The Free lunch is over“

Herb Sutter, 2005



Moore's Law

„Die Anzahl der Transistoren auf einem Chip verdoppelt sich alle 2 Jahre“



Dr. Pat Gelsinger, Sr. VP, Intel Corporation and GM, Digital Enterprise Group, February 19, 2004, Intel Developer Forum, Spring 2004

Manycore Shift

"After decades of single core processors, the high volume processor industry has gone from single to dual to quad-core in just the last two years. Moore's Law scaling should **easily let us hit the 80-core mark** in mainstream processors **within the next ten years** and quite possibly even less."
- Justin Ratner, CTO, Intel

WICHTIGE BEGRIFFE & KONZEPTE

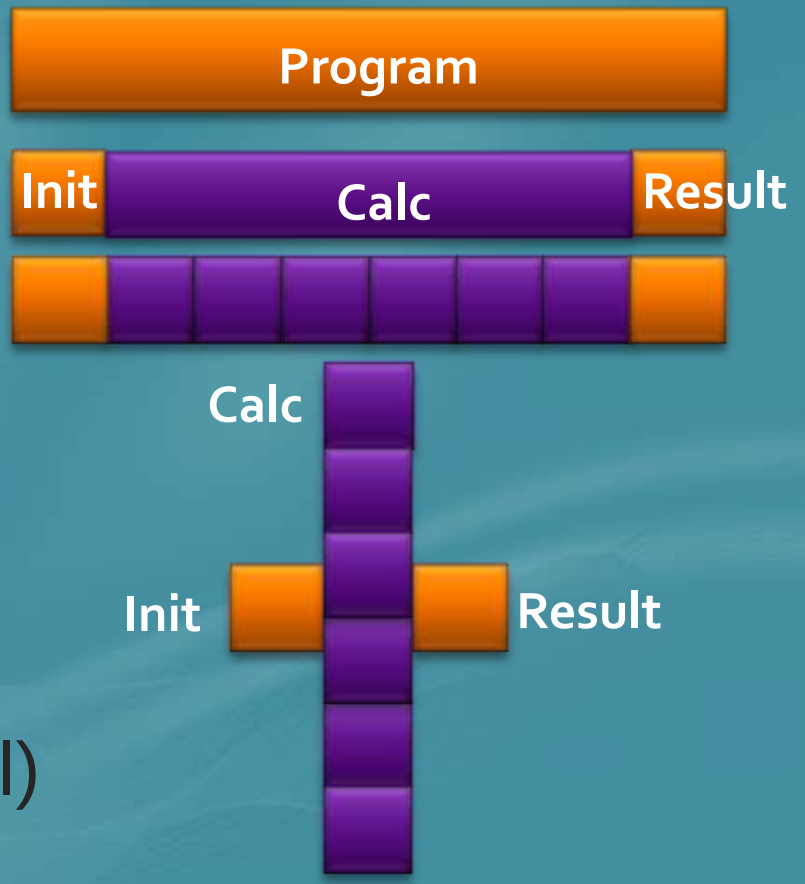
Microsoft Student Partners

Wichtige Begriffe

- Skalierbarkeit
  - Amdahl‘s Law
- Speedup

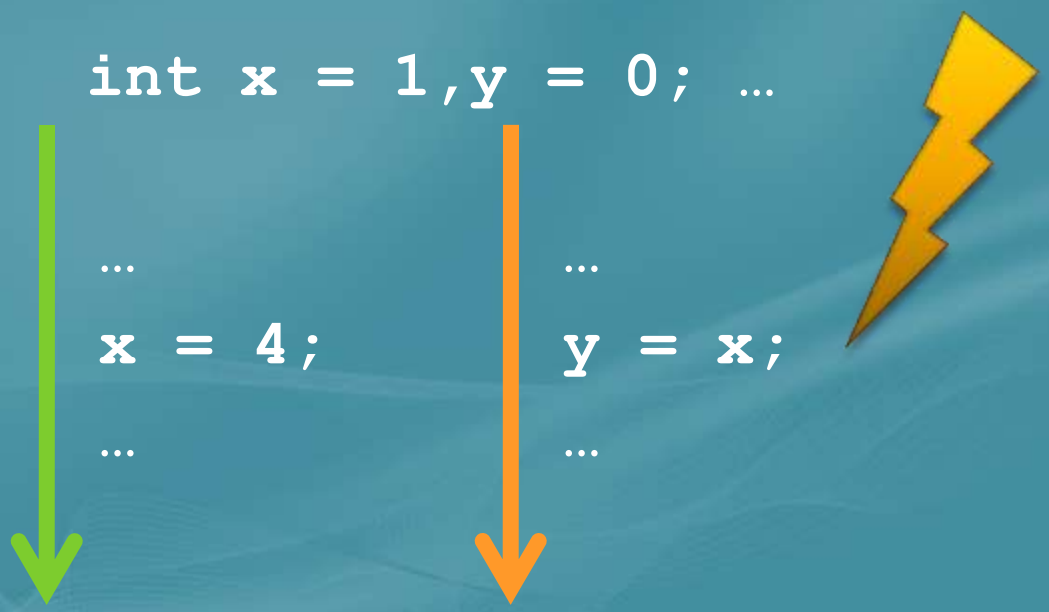
$$S(p) = \frac{T(1)}{T(p)}$$

  - < 1 Sublinear (Regelfall)
  - = 1 Linear (Ideal)
  - > 1 Superlinear (Selten)



Probleme: Data Race

- Zwei Threads greifen parallel auf dieselbe Variable zu, wobei **mindestens** einer der Zugriffe ein **Schreib** Zugriff ist.



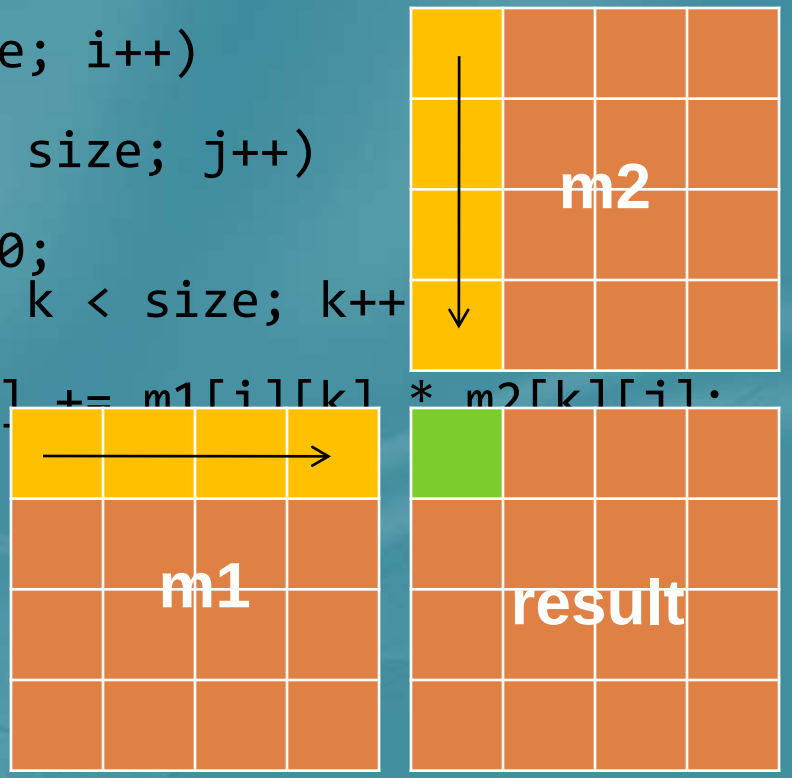
PROGRAMMIERPARADIGMEN

Microsoft Student Partners

NATIVER CODE - THREADS

Beispiel: Matrix Multiplikation

```
void MatrixMult(
    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];
            }
        }
    }
}
```



Parallele Matrix Mult. mit Threads

```
void MatrixMult
    int size, do                    ouble** m2, double** re
  int N = siz
  int P = 2 *
  int Chunk = N
  HANDLE hEven                      (NULL
  long counter = P;
  for (int c = 0; c < P;
    std::thread t ([&,c] {
      for (int i = c * Chunk;
           i < (c + 1 == P ? N :                    ) {
        for (int j = 0; j < size; j+
          result[i][j] = 0;
          for (int k = 0; k <
            result[i][j] +=
          }
        }
      }
      if (Interlo                    ounter) == 0)
        SetEve
    });
  }
  WaitForSingleOb       t(hEvent,INFINITE);
  CloseHandle(hEvent);
}
```

- Statische Aufteilung
- Manuelle Synchronisation
- Fehleranfällig
- Boilerplate Code
- Tricks
- Thread Wiederverwendung
- Manuelle Synchronisation

Microsoft Student Partners

Threads

- Vorteile:
  - Relativ wenig Overhead
  - Volle Kontrolle
- Nachteile:
  - Manuelle Synchronisation
    - Locks
    - Mutex
  - Manuelle Arbeitsverteilung auf Threads

➔Viel zusätzlicher Code nötig

NATIVER CODE - OPENMP

OpenMP

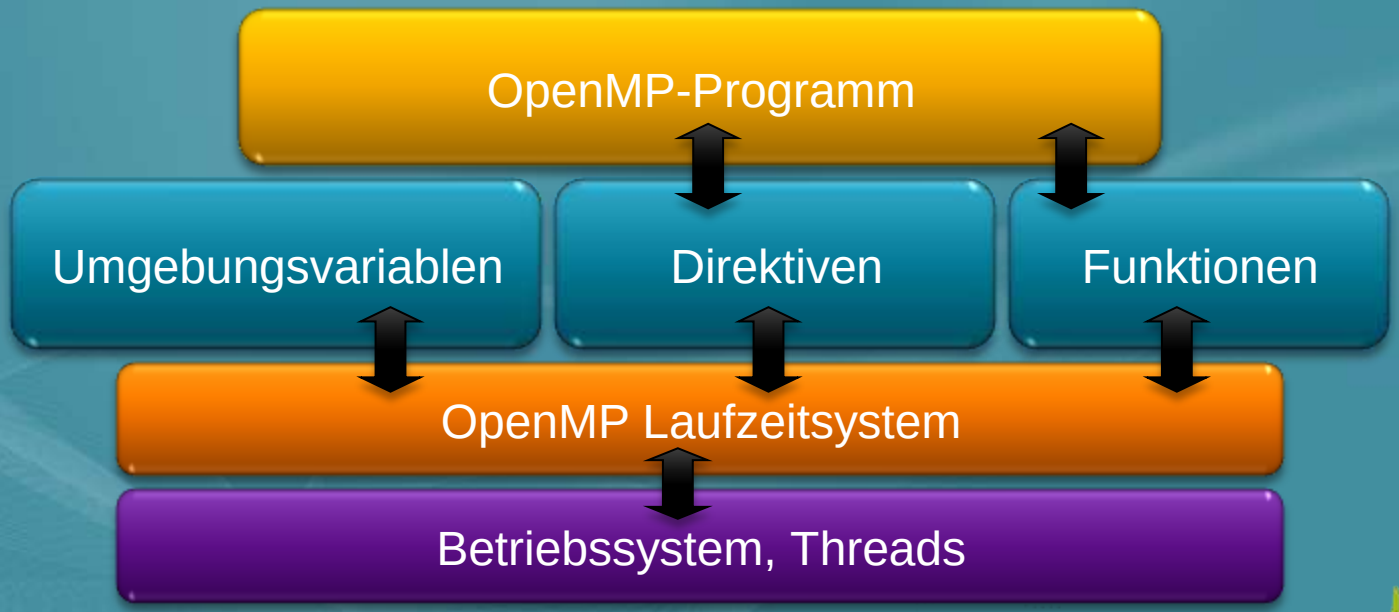
- **Open MultiProcessing**
- Standardisiert für Fortran und C/C++
  - aktueller Standard: 3.0
- MSFT: Support seit Visual Studio 2005
- Sonstige Compiler:
  - GCC, Intel, Sun, PGI

OpenMP

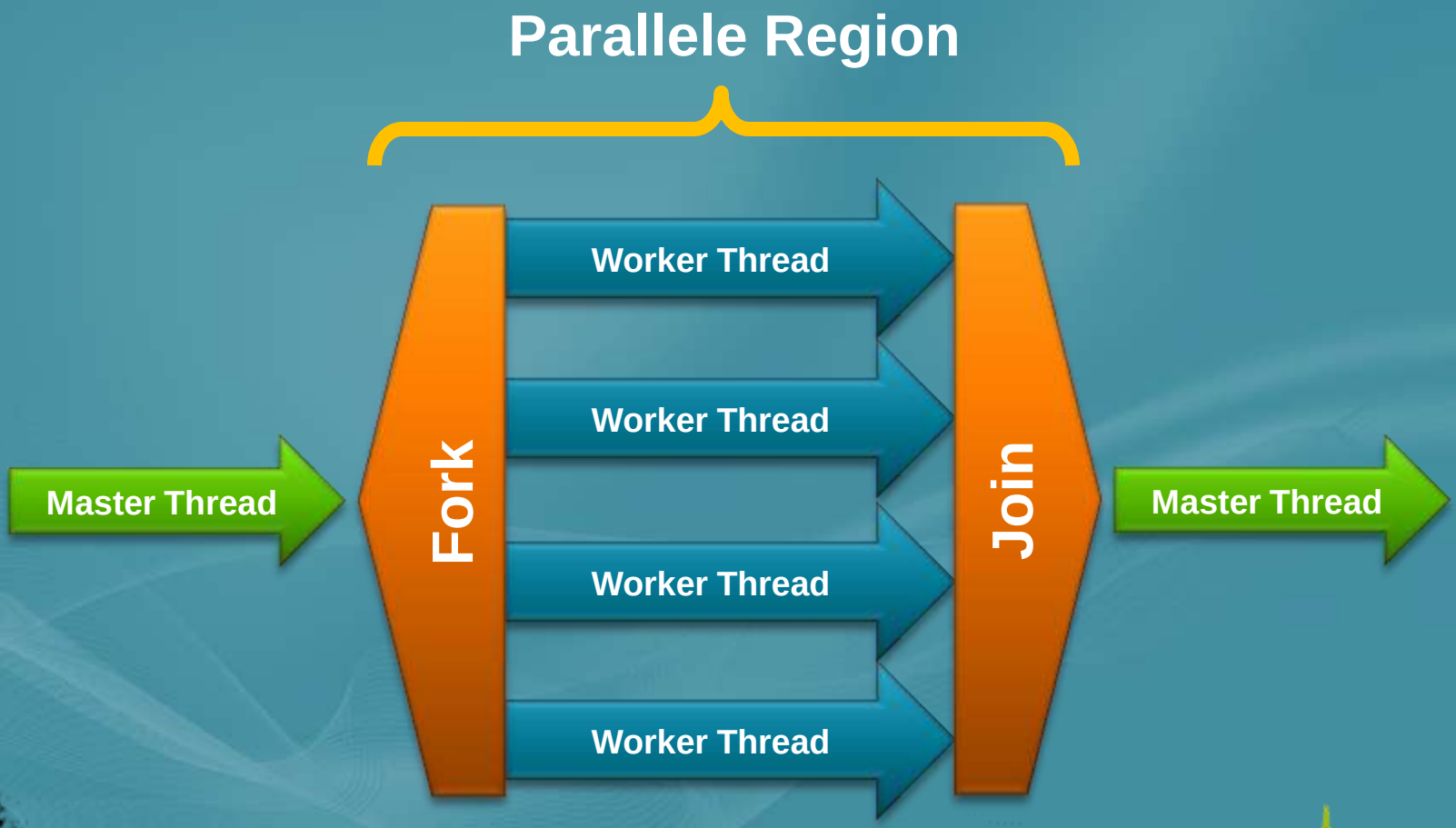
- *Fork-Join* Parallelismus über Direktiven
  - inkrementelle Parallelisierung möglich
- Hauptsächliches Einsatzgebiet:
  - **Data Parallelism** bzw „Loop-Level Parallelism“
  - High-Performance Computing
  - Wissenschaftliche Programme
  - Langwierige Berechnungen

Komponenten von OpenMP

- Umgebungsvariablen
- Direktiven
- Funktionen
- Laufzeitumgebung



OpenMP – Fork & Join



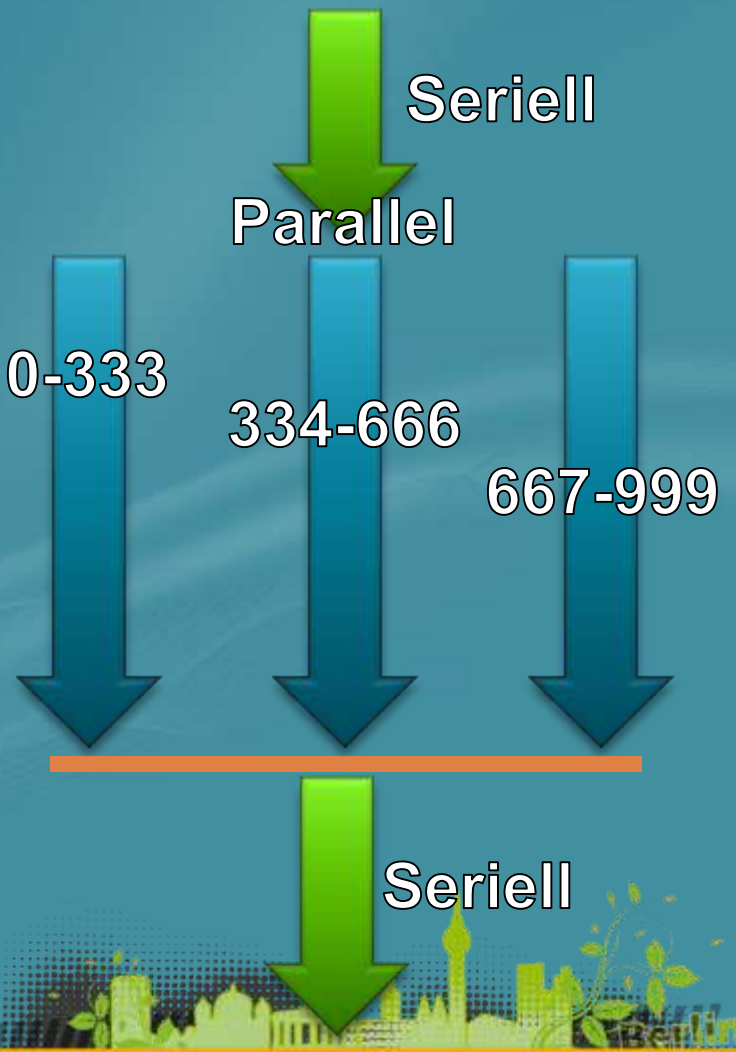
OpenMP – Beispiel

```
int main( int argc, char** argv )
{
   // serial code

#pragma omp parallel
   {
      // parallel execution
   }

   // serial code
}
```

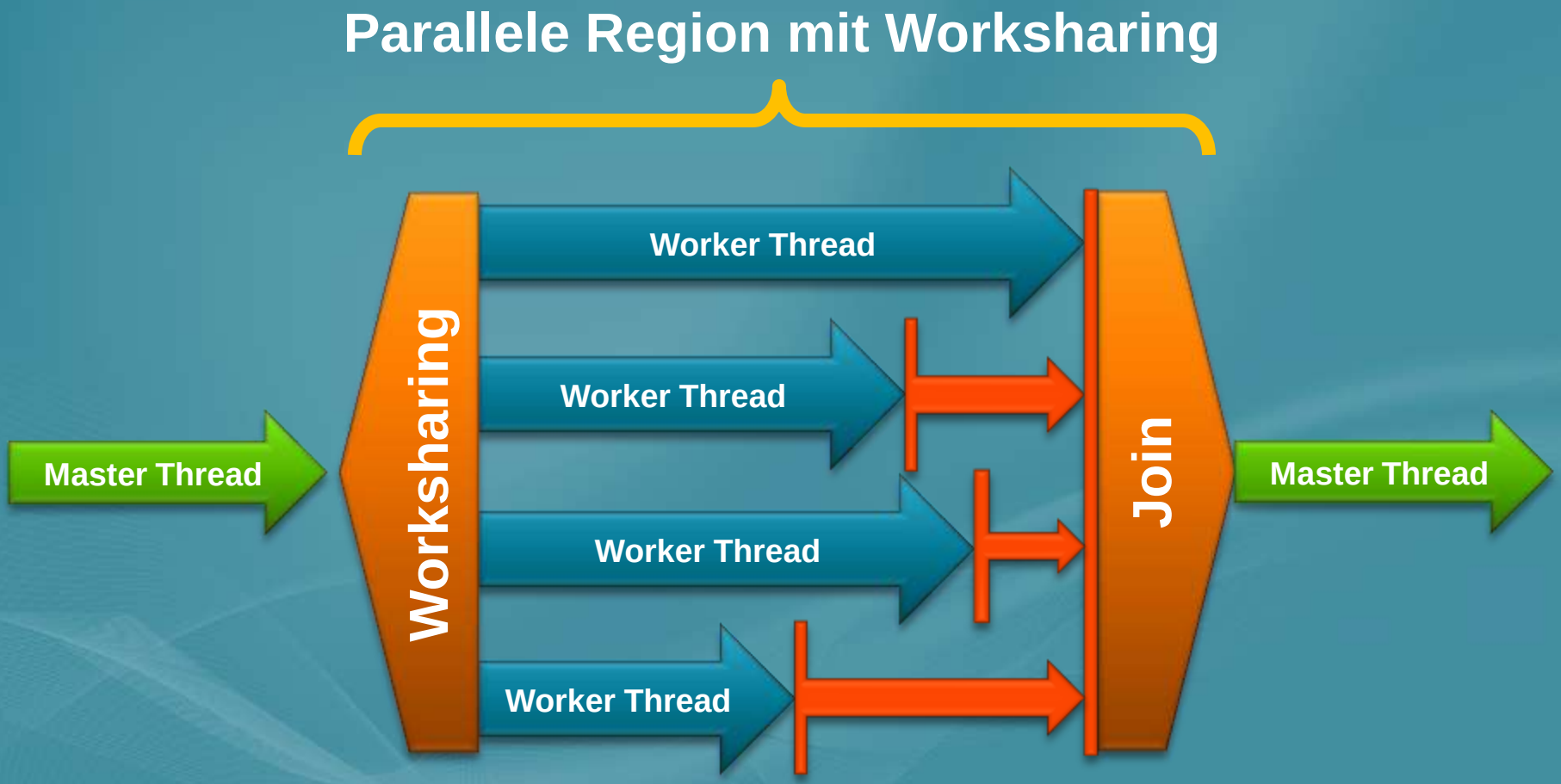
OpenMP – Worksharing

- Automatische Verteilung von Arbeit auf Threads

```
…
#pragma omp parallel for
            schedule(static)
for(i = 0; i < 1000; ++i)
{
  DoWork( i );
}
…
```



OpenMP - Schedule



OpenMP – Schedule

- Standard:
  `schedule(static,<chunksize>)`
- `schedule(dynamic,<chunksize>)`
- `schedule(guided,<chunksize>)`

OpenMP

- Weitere Direktiven:
  - Synchronisation

```
#pragma omp master
  { }

#pragma omp single
  {
     std::cout << "Hello from " <<
     omp_get_thread_num() << std::endl;
  }

#pragma omp barrier
```

WAS IST MIT .NET?

.NET Threads

- Wie nativ: Fork & Join
  - Threads einzeln verwalten

```
class Program
{
    static void Main(string[] args) {
        Thread t = new Thread(MachWas);
        t.Start("Hallo Welt!");
        t.Join();
        Console.WriteLine("Bearbeitung beendet");
    }
    static void MachWas(object o)     {
        Console.WriteLine((string)o);
    }
}
```

.NET Threads

- Vorteile – wie bei nativen Threads
  - Komplette Kontrolle über alle Threads
  - Wenig Overhead bei Erzeugung
- Nachteile – wie bei nativen Threads
  - Manuelle Erstellung und Arbeitsteilung
  - Load Balancing obliegt dem Programmierer
  - Manuelle Synchronisation und Verwaltung

.NET ThreadPool

- Threadverwaltung wird .NET überlassen

```
class Program
{
    static void Main(string[] args) {
        ThreadPool.QueueUserWorkItem(MachWas, "Hallo Welt!");
        ThreadPool.QueueUserWorkItem(MachWas, "Hallo Welt!");
        // something
        Console.WriteLine("Bearbeitung beendet");
    }
    static void MachWas(object o) {
        Console.WriteLine((string)o);
    }
}
```

.NET ThreadPool

- Vorteile
  - Zahl der Threads wird vom System automatisch gewählt
  - Threads werden wiederverwendet
  - Rudimentäres Load Balancing enthalten
- Nachteile
  - Aufgabenteilung manuell
  - Data Races immer noch möglich

.NET 4.0 PARALLEL EXTENSIONS

Microsoft Student Partners

.NET 4.0 - Tasks

- Power of Thread & Elegance of ThreadPool
- Was sind die Vorteile von Tasks?
  - Automatische Anpassung der Threadzahl
  - Man definiert Aufgaben, nicht Threads
  - Work-Stealing Algorithmus zur Vermeidung von Load Imbalance
- Was sind die Nachteile?
  - Etwas mehr Overhead als bei Threads

.NET 4.0 - Tasks

```
class Program
{
   static void Main(string[] args) {
      Task t = Task.Factory.StartNew(delegate {
                  MachWas("Hallo Welt!")} );
      t.Wait();
      Console.WriteLine("Bearbeitung beendet");
   }
   static void MachWas(object o){
      Console.WriteLine((string)o);
   }
}
```

Task Parallel Library (TPL)

- Statische Klasse: **`Parallel`**
  - Automatische Parallelisierung von For/ ForEach und deren generischen Versionen
  - **`Invoke`** zur Ausführung als asynchroner Task

Task Parallel Library (TPL)

```
class Program
{
    static void Main(string[] args) {
        Parallel.For(0, iNumIter, i =>
        {
            MachWas("Hallo Welt!");
        });
        Console.WriteLine("Bearbeitung beendet");
    }
    static void MachWas(object o)     {
        Console.WriteLine((string)o);
    }
}
```

PLINQ

- Parallel LINQ (PLINQ)
  - LINQ Queries parallel auszuführen
  - Automatische Parallelisierung der Anfragen mit AsParallel()

```
var q = from p in people.AsParallel()
        where p.Name == queryInfo.Name &&
              p.State == queryInfo.State &&
              p.Year >= yearStart &&
              p.Year <= yearEnd
        orderby p.Year ascending
        select p;
```

PLINQ

```
class Program
{
  static void Main(string[] args) {
    IEnumerable<int> numbers = Enumerable.Range(1, 1000000);
    IEnumerable<int> primes = from n in
                              numbers.AsParallel()
                              where isPrime(n) select n;
    Console.WriteLine("Bearbeitung beendet");
  }
  static bool IsPrime(int n) {
    // test if prime
  }
}
```

Q&A

Microsoft Student Partners

Links

- MSDN Portal zum Thema Parallel Computing
  http://msdn.microsoft.com/concurrency
- Deutsche Windows HPC Usergroup
  www.rz.rwth-aachen.de/li/k/sbb

Microsoft®

Your potential. Our passion.™

Microsoft Student Partners