Multi- und Many-Core

Benjamin Warnke

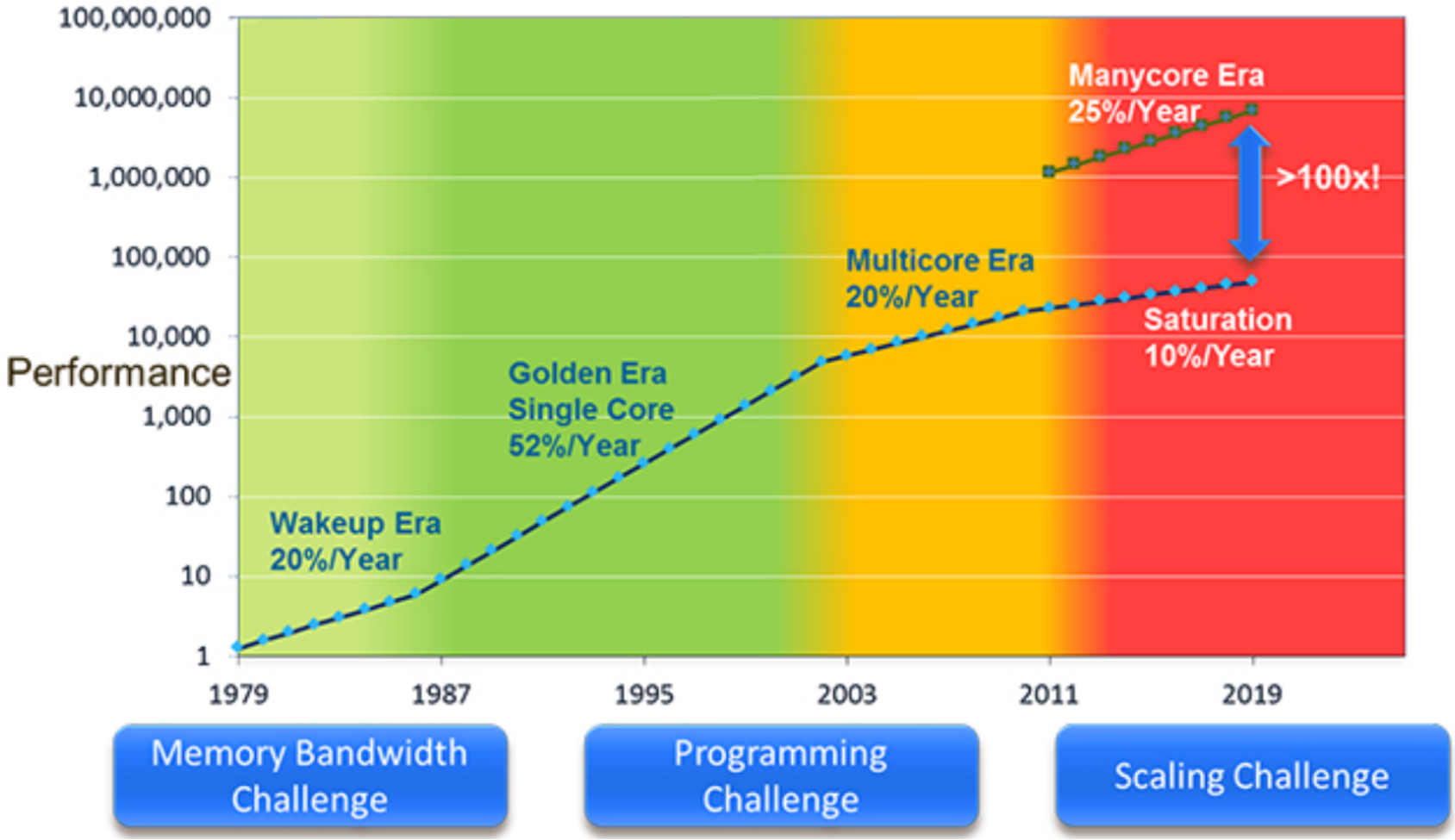
Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2016-12-15

Gliederung (Agenda)

Gliederung (Agenda)

Herausforderungen im Verlauf der Zeit



verändert nach [1]

Warum Multi- und Manycore

Ziel: Mehr Rechenleistung (Flops/sec)
erster Ansatz: höherer Takt
→ höhere Spannung
→ mehr Hitze
→ unmöglich zu kühlen
neuer Ansatz: langsamer Takt dafür mehr gleichzeitig
→ kleinere Spannung
→ Energieeffizienz

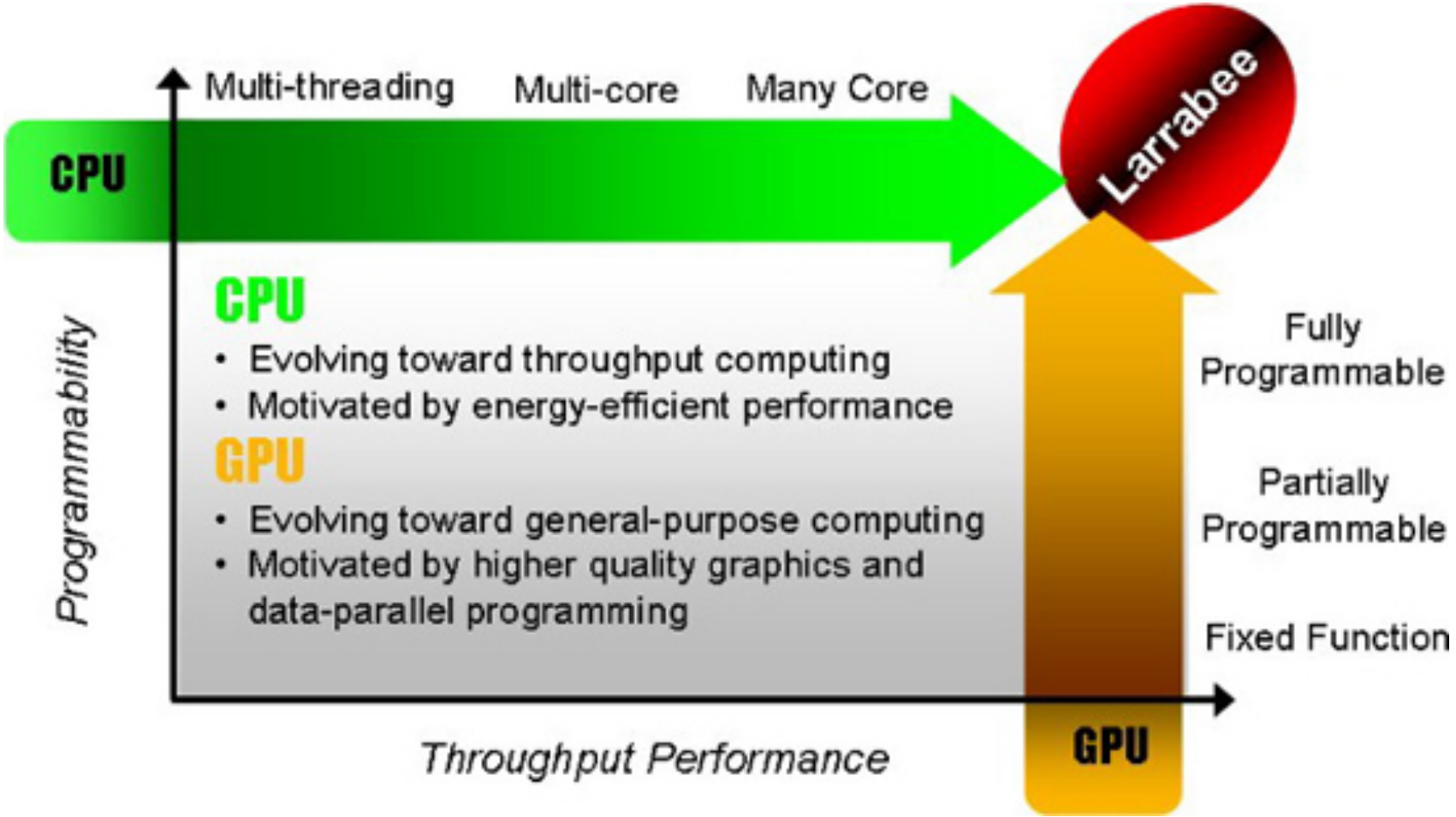
Probleme von Mehrkernsystemen

- Speicherbandbreite
- erschwertes Debuggen
- Algorithmus muss parallelisierbar sein
  → nicht jede Parallelisierung ist schneller
- Nichtdeterminismus
- Overhead durch Parallelisierung
- Synchronisierung → Sperr-Verfahren → deadlocks

Multi- und Manycore

- Systemprozessor (2 – 16 Kerne) → Multicore
  - POSIX-Threads
  - OpenMP
  - TBB
  - MPI
  - OpenACC
- GPU (> 1000 Kerne) → Manycore
  - CUDA
  - OpenCL
  - OpenGL (für Grafikberechnungen)
  - OpenACC
- Xeon Phi (~ 70 Kerne) → Manycore
  - Mischform

Xeon Phi - Mischform



Quelle: [5]

Xeon Phi

- (Co-)Prozessor
- PCI-Express/Sockel
- ~ 70 Kerne
- ~ 280 Threads
- langsamer Takt
- Hardware optimiert für parallelen Code
- besonders gut geeignet für Vektoroperationen
- (nur PCI-Express) verschiedene Modi
  - Rechenlast aufteilen mit Systemprozessor
  - Eigenenständiger Rechnerknoten
  - Wie GPU nur verwendet bei Bedarf durch CPU
- unterstützt MPI

Thread vs Prozess

- Gemeinsam
  - Unabhängige parallele Code-Ausführung
  - vom Betriebssystem verwaltet
- Prozess
  - eigener Adressraum im Speicher
  - Kommunikation durch Nachrichten/ Dateien/ Shared Memory Segmente
  - im Fehlerfall kein unmittelbarer Einfluss auf andere Prozesse
- Thread
  - gemeinsamer Speicher
  - innerhalb eines Prozesses
  - wenn 1 Thread abstürzt, dann auch der gesamte Prozess

Gliederung (Agenda)

Thread Parallelisierung

- Vorteile
  - kein Kopieren der Daten notwendig
  - bessere Zugriffszeit
  - Compiler können bei Datenaufteilung helfen
  - Compilergestützter Lastausgleich möglich
- Nachteile
  - gemeinsame Speicherbandbreite

Thread Building Blocks

C++ template library

- Stärken
  - Parallele Datentypen z.B. "vector", "queue", "map"
  - nicht balancierte Schleifen
  - work stealing
  - starke Compiler Optimierung
  - zur Laufzeit anpassbar
  - gute Code Lesbarkeit durch Templates
  - Open Source
- Schwächen
  - nur C++
  - läuft auf diesem Cluster langsam

Schleifen Parallelisierung

```
1  #include "tbb/tbb.h"
2  void SerialLoop(float a[]) {
3      for(int i = 0; i < a.length; i++){
4          DoSomething(i,a[i]);
5      }
6  }
7  void ParallelLoop(float a[]) {
8      tbb::parallel_for(0, a.length, [&](int i) {
9          DoSomething(i,a[i]);
10     });
11 }
```

Listing 1: Schleifen Parallelisierung

Queue Parallelisierung

```
1  #include "tbb/tbb.h"
2  std::queue<T> MySerialQueue;
3  tbb::concurrent_queue<T> MyParallelQueue;
4  T item;
5  void SerialPop(){
6      if(!MySerialQueue.empty() ) {
7          item = MySerialQueue.front();
8          MySerialQueue.pop_front();
9          DoSomething(item);
10     }
11 }
12 void ParallelPop(){
13     if(MyParallelQueue.try_pop(item) ) {
14         DoSomething(item);
15     }
16 }
```

Listing 2: Queue Parallelisierung

OpenMP

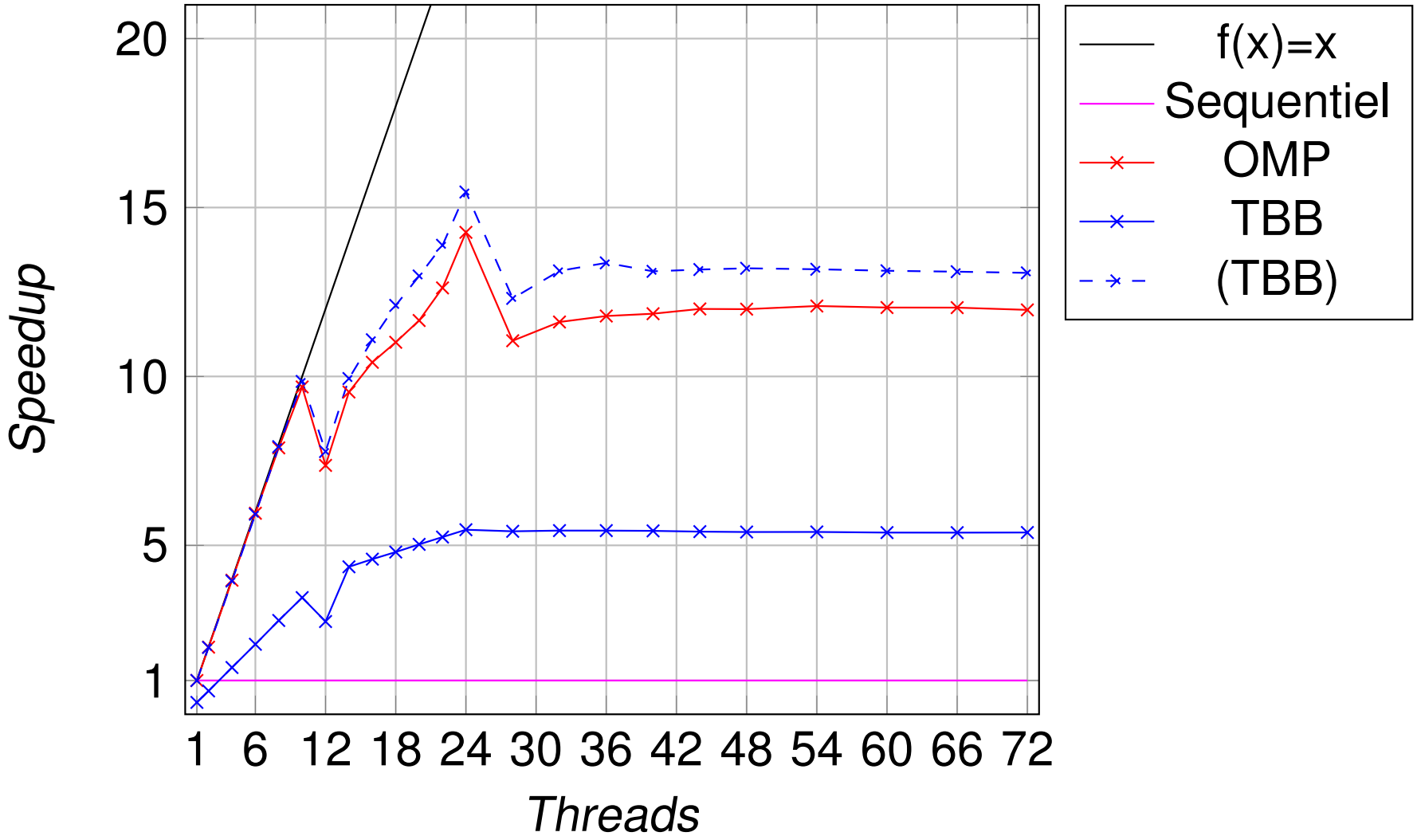
- Stärken
  - Parallelisierung durch Compiler Anweisungen
  - iterative Parallelisierung möglich
  - Vektorisierung wird nicht beeinflusst
- Schwächen
  - keine parallelen Datenstrukturen enthalten
  - nicht alle Compiler unterstützen OpenMP
  - erhöhte Fehlergefahr durch Variablen, die private sein sollten

Schleifen Parallelisierung

```
1  void SerialLoop(float a[]) {
2      for(int i = 0; i < a.length; i++){
3          DoSomething(i,a[i]);
4      }
5  }
6  void ParallelLoop(float a[]) {
7  #pragma omp parallel for
8      for(int i = 0; i < a.length; i++){
9          DoSomething(i,a[i]);
10     }
11 }
```

Listing 3: Schleifen Parallelisierung

partdiff-xxx.exe $i 2 1024 2 2 800



Gliederung (Agenda)

Prozess Parallelisierung

- Vorteile
  - beliebig viele Prozesse
  - größere Speicherbandbreite
  - größerer (Haupt-) Speicher
- Nachteile
  - Kopieren von Datenteilen notwendig
  - Compiler können bei Optimierung kaum helfen
  - Lastausgleich muss explizit codiert werden
  - nur wenige und teure Debugger
  - sehr aufwendiges Debuggen

Message Passing Interface

Bibliothek für Nachrichtenaustausch

- Stärken
  - beliebig viele Prozesse
  - verteilte Systeme (Cluster)
  - Implementierungen für C, C++, Fortran, Java, Python, $\cdots$
- Schwächen
  - expliziter Code notwendig
  - explizite Datenaufteilung
  - benötigt wrapper Compiler

Hello World

```
1  #include <mpi.h>
2  int world_size;// Anzahl der Prozesse
3  int world_rank;// Nummer dieses Prozesses
4  int main(int argc, char** argv) {
5      MPI_Init(&argc,&argv);
6      MPI_Comm_size(MPI_COMM_WORLD, &world_size);
7      MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
8      doSomething();
9      MPI_Finalize();
10 }
```

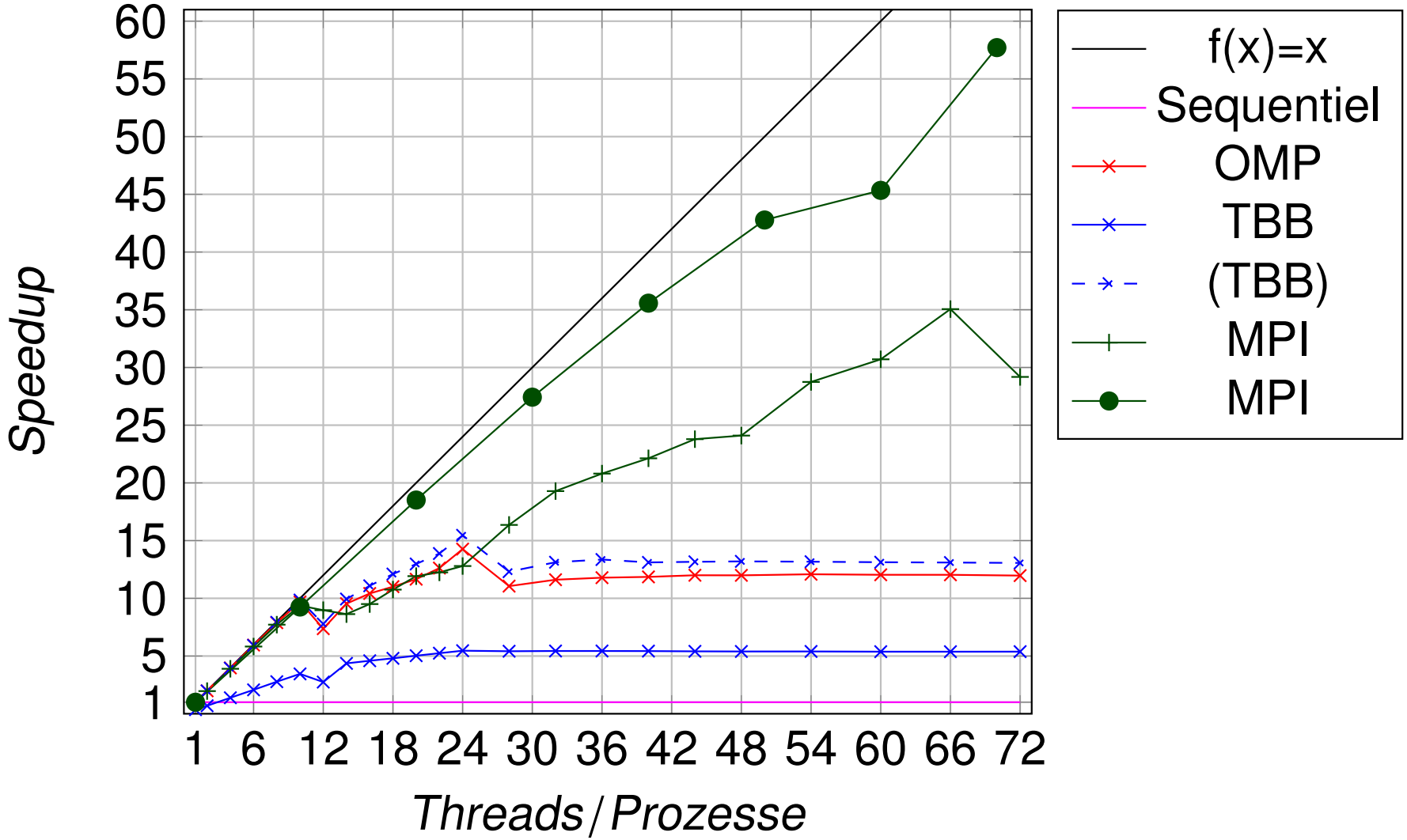
Listing 4: Hello World

Hello World

```
1  #include <stdio.h>
2  #include <mpi.h>
3  void doSomething() {
4      printf("Hello World from Rank %d \n", world_rank);
5      int number;
6      if (world_rank == 0) {
7          number = rand();
8          MPI_Send(&number, 1, MPI_INT, 1, 0,
               ↪ MPI_COMM_WORLD);
9      } else {
10         MPI_Recv(&number, 1, MPI_INT, 0, 0,
               ↪ MPI_COMM_WORLD, MPI_STATUS_IGNORE);
11         printf("received %d \n", number);
12     }
13 }
```

Listing 5: Hello World

partdiff-xxx.exe $i 2 1024 2 2 800

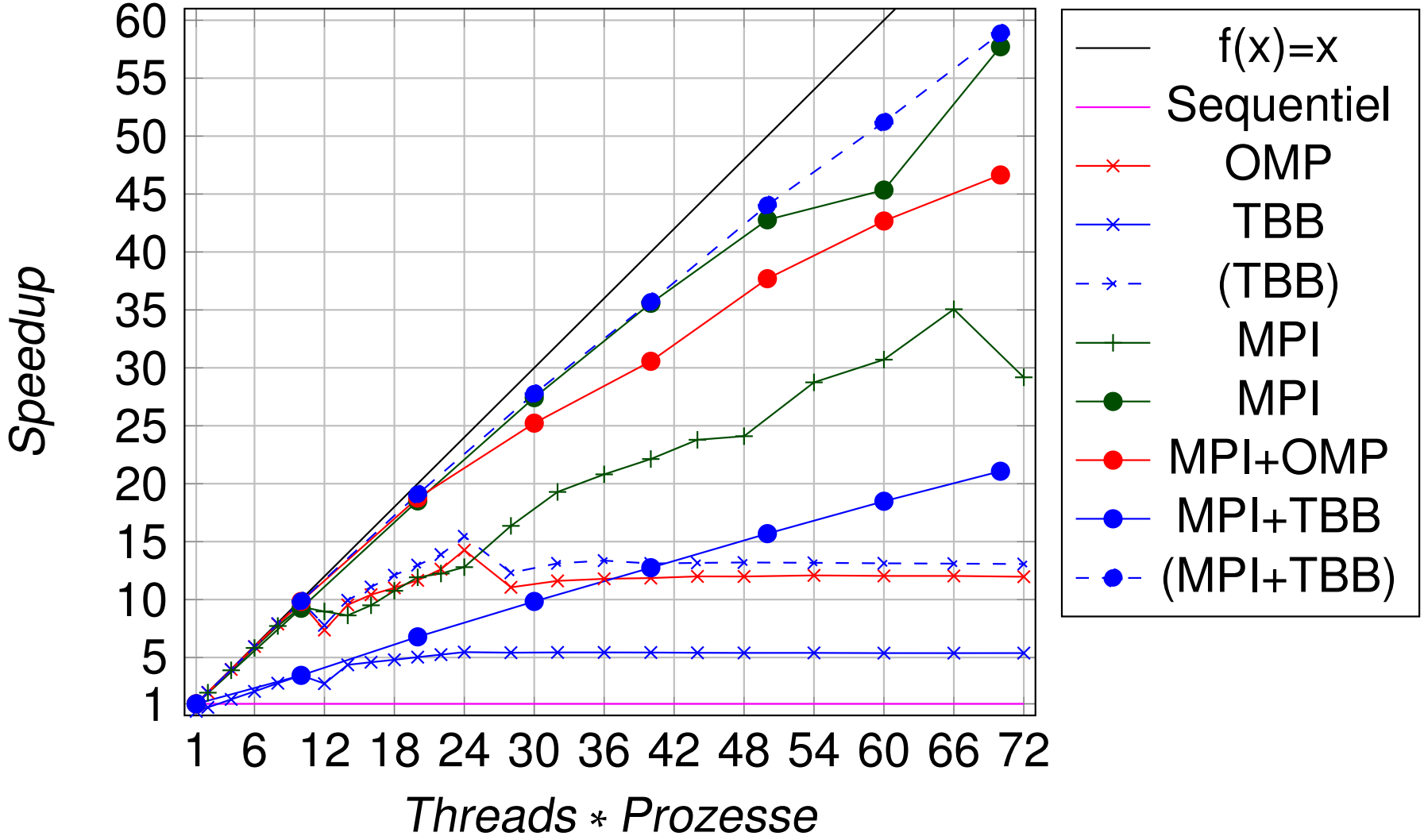


Gliederung (Agenda)

Kombination Threads und Prozesse

- Vorteile
  - gemeinsamer Cache für Threads
  - größere Speicherbandbreite durch mehrere Rechenknoten
- Nachteile
  - Kopieren von Daten über das Netzwerk nötig
  - Lastausgleich muss explizit codiert werden
  - Fehlerquellen von Threads + Prozessen
  - sehr aufwendiges Debuggen

partdiff-xxx.exe $i 2 1024 2 2 800



Gliederung (Agenda)

Zusammenfassung

- Probleme von Multicore-Architekturen
- Xeon Phi (Co-)Prozessor
- TBB und OpenMP
- MPI zur Kommunikation zwischen verschiedenen Rechenknoten
- verschiedene Varianten der Parallelisierung sind kombinierbar

Gliederung (Agenda)

Literatur I

[1] “Adapteva turns to Kickstarter to fund massively parallel processor.” [Online]. Available: https://www.extremetech.com/electronics/137085-adapteva-turns-to-kickstarter-to-fund-massively-parallel-p

[2] “Die Evolution von Multi-Core- zu Many-Core-Prozessoren.” [Online]. Available: ftp://ftp.ni.com/pub/branches/germany/2015/artikel/06-juni/04_Der_Kern_der_Materie_Tom_Bradicich_Invision_3_2015.pdf

[3] “Manycore processor.” [Online]. Available: https://en.wikipedia.org/wiki/Manycore_processor

Literatur II

[4] “What is the difference between many core and multi core?” [Online]. Available: http://electronics.stackexchange.com/questions/37982/what-is-the-difference-between-many-core-and-multi-core

[5] “The Future of Larrabee: The Many Core Era.” [Online]. Available: http://www.anandtech.com/show/2580/14

[6] “Ask James Reinders: Multicore vs. Manycore.” [Online]. Available: https://goparallel.sourceforge.net/ask-james-reinders-multicore-vs-manycore/

[7] “Xeon Phi.” [Online]. Available: http://www.intel.de/content/www/de/de/processors/xeon/xeon-phi-detail.html

Literatur III

[8] “Intel® Threading Building Blocks, OpenMP, or native threads?” [Online]. Available: https://software.intel.com/en-us/intel-threading-building-blocks-openmp-or-native-threads

[9] “Intel® Threading Building Blocks (Intel®TBB) Developer Guide.” [Online]. Available: https://software.intel.com/en-us/tbb-user-guide

[10] “Generic Parallel Algorithms.” [Online]. Available: https://www.threadingbuildingblocks.org/tutorial-intel-tbb-generic-parallel-algorithms

Literatur IV

[11] “Concurrent Containers.” [Online]. Available: https://www.threadingbuildingblocks.org/tutorial-intel-tbb-concurrent-containers

[12] “Message Passing Interface.” [Online]. Available: https://de.wikipedia.org/wiki/Message_Passing_Interface

[13] “MPI Hello World.” [Online]. Available: http://mpitutorial.com/tutorials/mpi-hello-world/