Towards a Performance Engineering Workflow for OpenMP 4.0

TU Darmstadt:

Christian M. Iwainsky

Prof. Christian H. Bischof

RWTH Aachen University

Christian Terboven

Dirk Schmidl

Prof. Matthias S. Müller

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Multi-Core Systems Common**
- **OpenMP Attractive Solution**
  - Incremental parallelization
  - No complete rewrite of the code
  - Easy benefit from multi-core systems
- **Parallel Programming Expertise Cumbersome to Obtain**
- **Need: Step by Step Guide-Lines for Good OpenMP Performance**
  - Distilled our OpenMP course experience
- **Test Approach on a CG-kernel**

Agenda

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Motivation
2. OpenMP Course Contents
3. CG-Solver Kernel
4. Workflow
5. Verification
6. Conclusion

OpenMP Workshop Contents

TECHNISCHE UNIVERSITÄT DARMSTADT

1) **Basic Computer Architecture (½ Day)**

   E.g.: Memory Hierarchy, NUMA, OS-Features for NUMA Support, Typical Performance Problems

2) **OpenMP Language Tutorial (3/4 Day)**

   E.g.: Basic Concept of Parallel Regions, Worksharing, Scoping, Memory Model, Synchronization, Tasking

3) **Optimization and Tuning Aspects of OpenMP Programs (3/4 Day)**

   E.g.: Thread Binding, NUMA Memory Allocation, Dynamic & Custom Scheduling, etc.

4) **Basic Introduction to Performance Analysis (1 Day)**

   E.g.: Hotspot Identification, Analysis of Thread Utilization

The CG Algorithm for Sparse Matrices

$$r_0 = b - \boldsymbol{A}x_0$$
$$d_0 = r_0$$

1 Sparse MatriX Vector (SMXV)
2 Vector Operations:
- Norm
- Dot-Product

1 Memory Copy

$$for\ (k = 0\ ..\ k_{max})$$

$$z = \boldsymbol{A}d^k$$
$$\alpha_k = \frac{r_k^T - r_k}{d_k^T z}$$
$$x_{k+1} = x_k - \alpha_k d_k$$
$$\beta_k = \frac{r_{k+1}^T - r_{k+1}}{r_k^T r}$$
$$d_{k+1} = r_{k+1} + \beta_k d_k$$
$$until\ \|r_{k+1}\| < \text{tol}$$

1 SMXV
5 Vector Operations:
- Dot-Product
- Scaled-Vector-Addition

SMXV

```
for(i=0; i<n; i++)
{
   y[i]=0;
   for(j=ptr[i]; j<ptr[i+1]; j++)
   {
      y[i]+=value[j]*x[index[j]];
   }
}
```

First Draft for Performance Engineering Workflow

TECHNISCHE UNIVERSITÄT DARMSTADT

Classic OpenMP Workflow

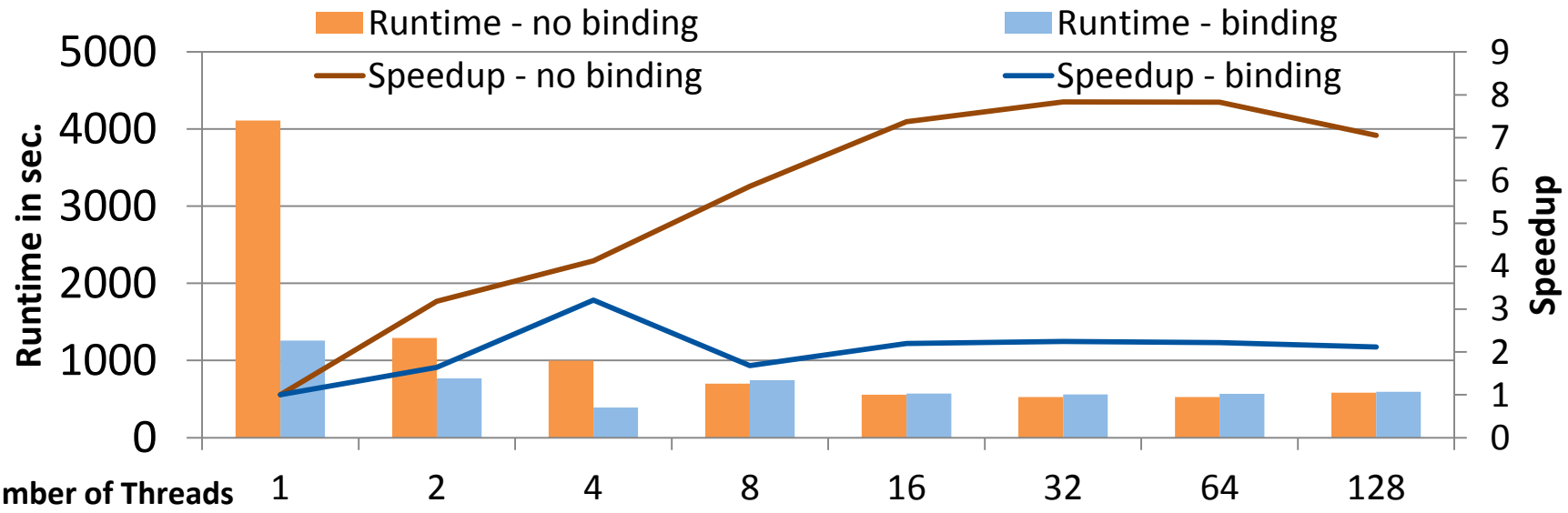
1) Naive Parallelization
2) Extend Parallel Regions
3) Add NUMA Awareness
4) Implement Load Balancing

} Use Performance Analysis Tools

Naive Parallelization

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Create Profile**
- **Parallelize all Hotspots (e.g. Loopnests) Separately**
- **Enable Thread Binding**
- **Target System: 16-Socket Nehalem-EX System with BCS**

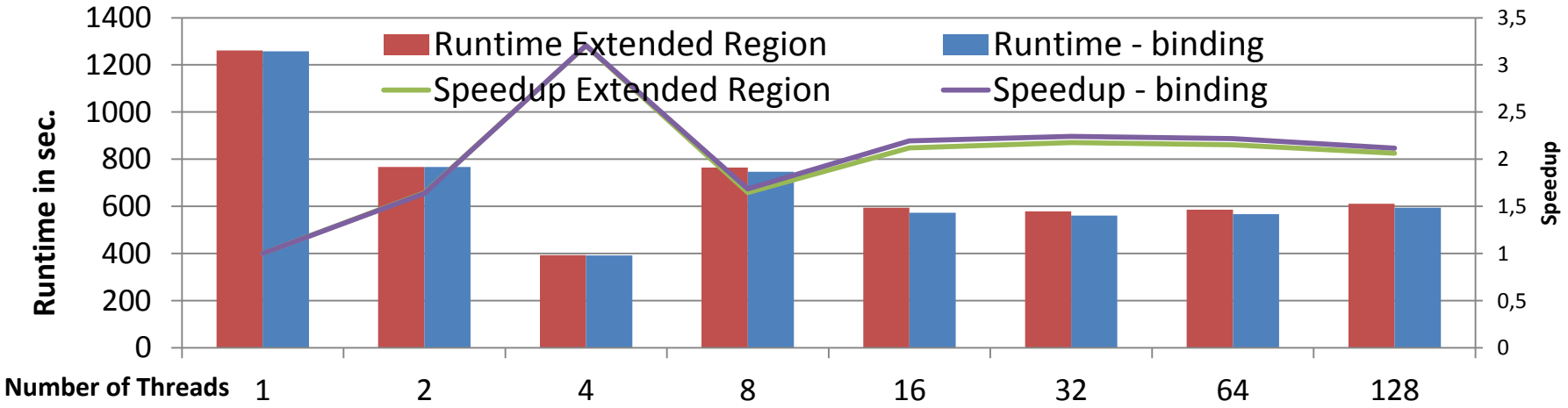


SCIENTIFIC COMPUTING

Extended Parallel Regions

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Extend all Parallel Regions to create a Single Parallel Region**



- **Overhead of Open Constructs**

OpenMP Construct	Overhead / Construct @128 Threads
Parallel For	660 µs
For	360 µs

Naive Implementation: 6 Parallel For Constructs - > only 1500 µs Saving per Iteration

SCIENTIFIC COMPUTING

NUMA Awareness

TECHNISCHE UNIVERSITÄT DARMSTADT

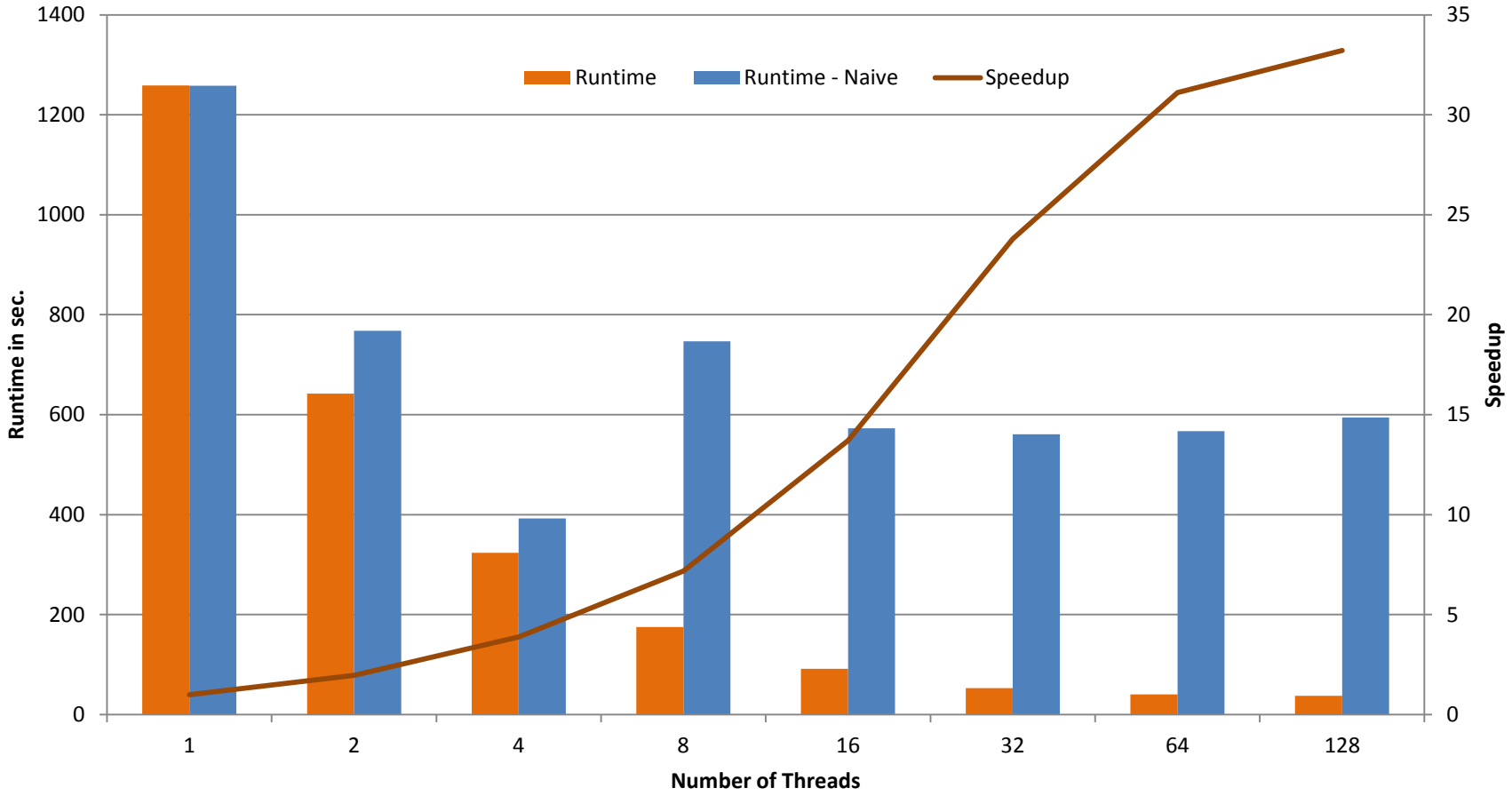
- **Performance Measurement with PAPI Counters (4 Socket Setup)**

Counter	Memory Access Percentage
MEM_UNCORE_RETIRED:LOCAL_DRAM	25.65%
MEM_UNCORE_RETIRED:REMOTE_DRAM	74.35%

➡ Data Locality Issue

- **Initialize all data structures on the NUMA node used by its thread**

NUMA Awareness: Result



Load Balancing

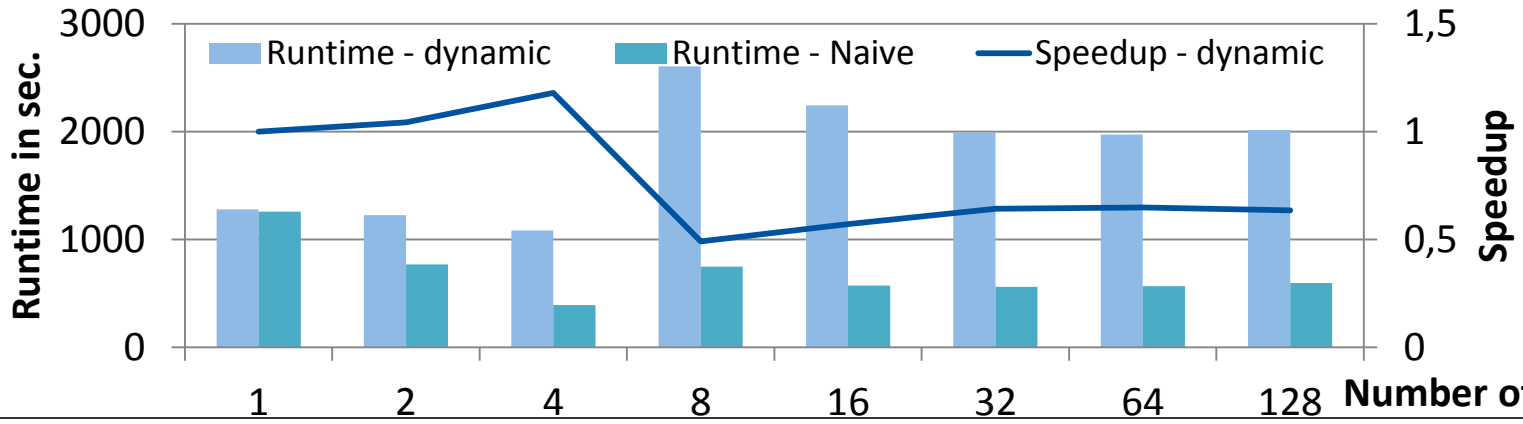
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Performance Measurement (Score-P Profile)**

Subroutine	OpenMP Waittime @ Barrier
Sparse Matrix Vector Multiplication	196 s
Vector Addition	2 s

➡ Load Balancing Issue in SMXV

- **Simple solution by using dynamic loop scheduling**



SCIENTIFIC COMPUTING

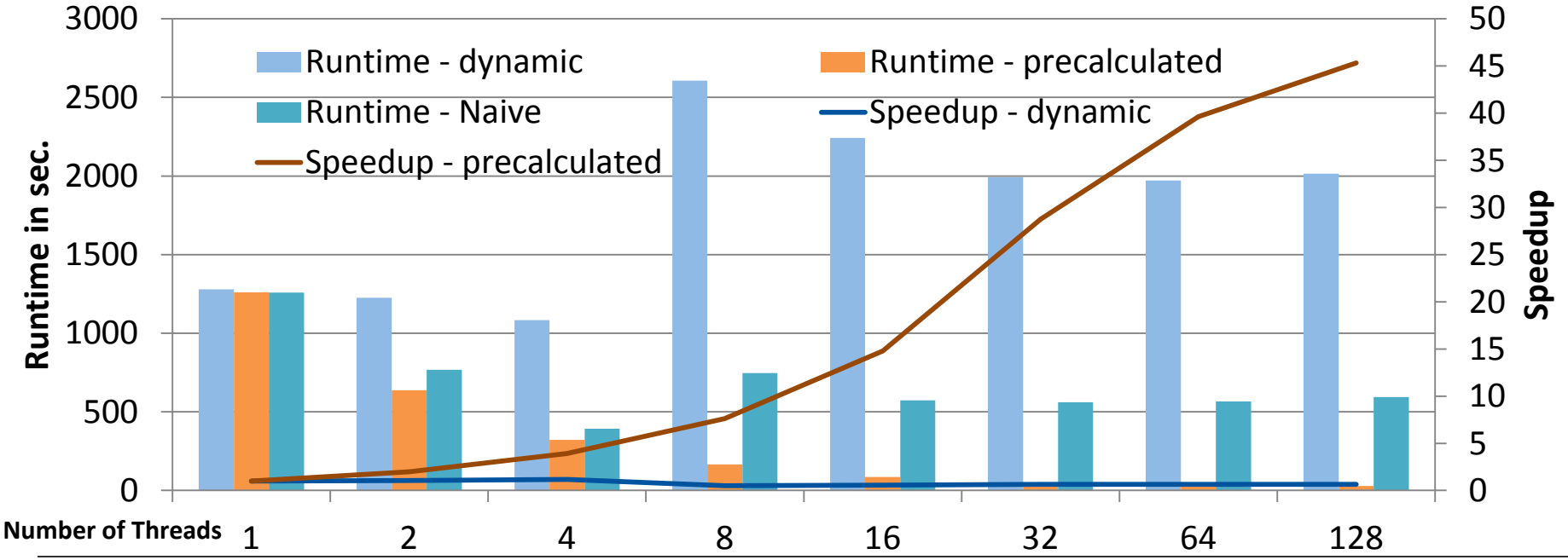
Load Balancing and NUMA revisited

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Performance Measurement showed increase in remote memory access: 70% remote accesses**

  ⇨ NUMA Aware Memory Initialization fails to match processing by threads

- **Solution: Manual, precalculated thread-distribution for SMXV**



SCIENTIFIC COMPUTING

First Draft for Performance Engineering Workflow

TECHNISCHE UNIVERSITÄT DARMSTADT

Pure OpenMP Workflow

1) Naive Parallelization
2) Extend Parallel Regions
3) Add NUMA Awareness
4) Implement Load Balancing

} Use Performance Analysis Tools

Coprocessor Workflow

a) Naive Coprocessor Parallelization
b) Minimize Data Transfer
c) Extend Parallel Regions
d) Add NUMA Awareness
e) Extend Target Region

} Use Performance Analysis Tools

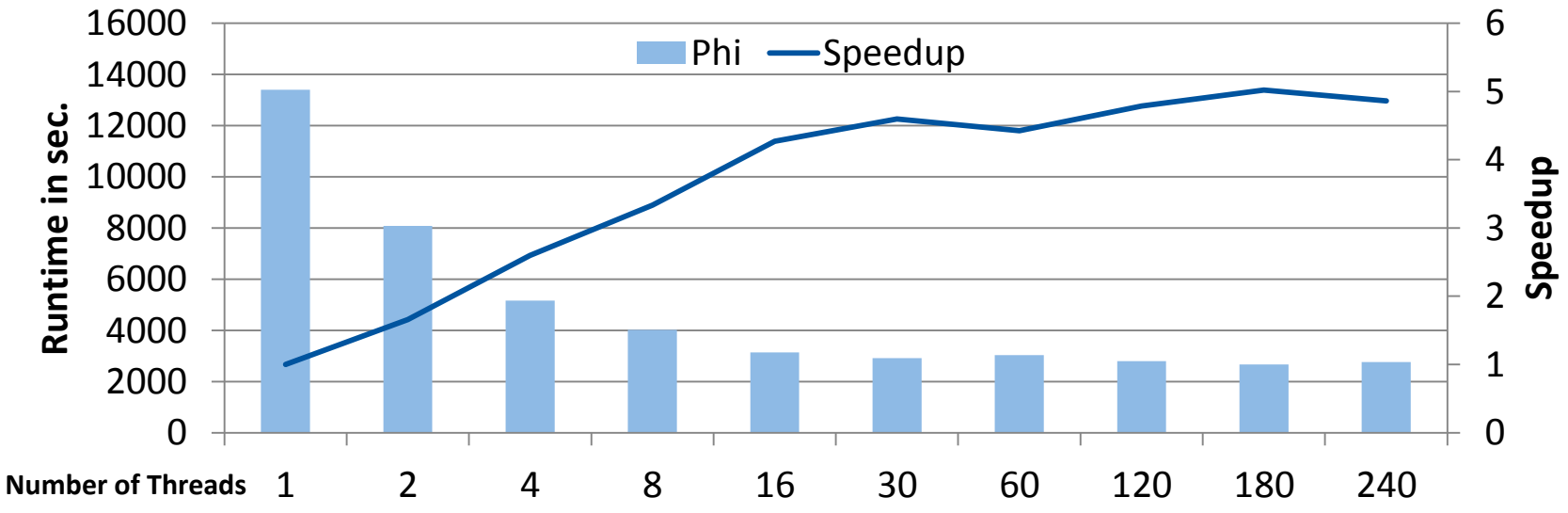
Coprocessor Approach

TECHNISCHE UNIVERSITÄT DARMSTADT

- **OpenMP 4.0 Target Device Regions**
- **Naive Implementation**

  Individual Hotspots as target-regions
- **Target System: Intel Xeon Phi (Xeon Phi 5110P)**

  Serial version very slow in comparison to the host CPU

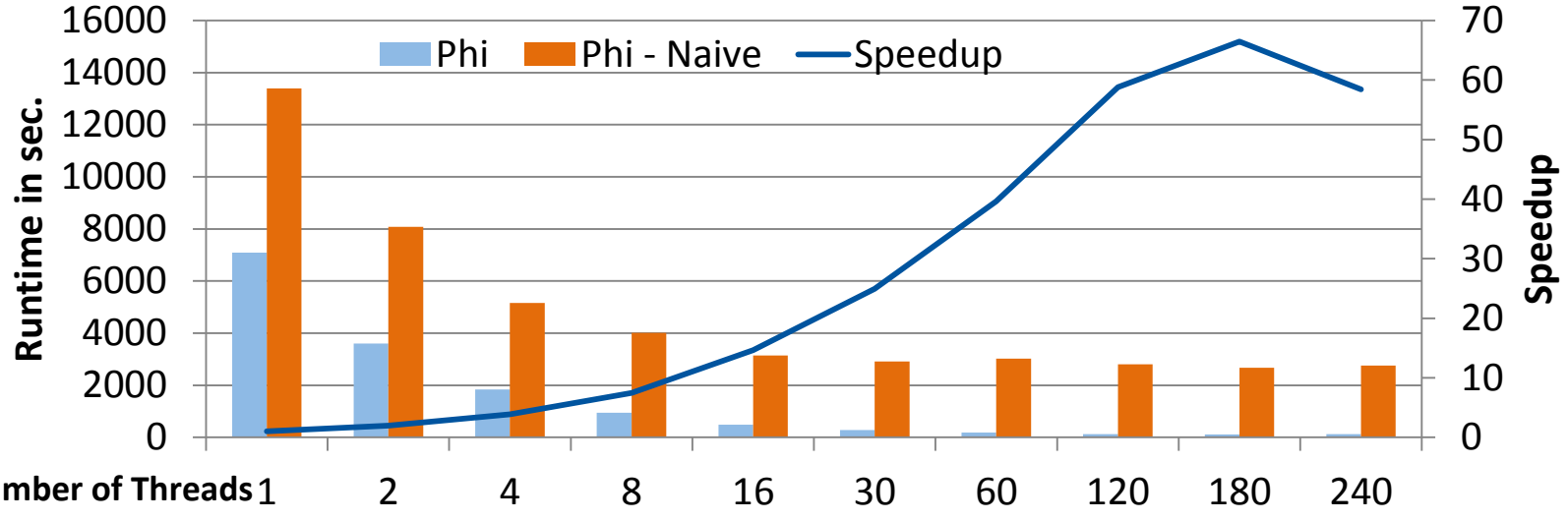


SCIENTIFIC COMPUTING

Minimizing Data Transfer

- **Accelerator Transfer Investigation** (OFFLOAD_REPORT = 2)

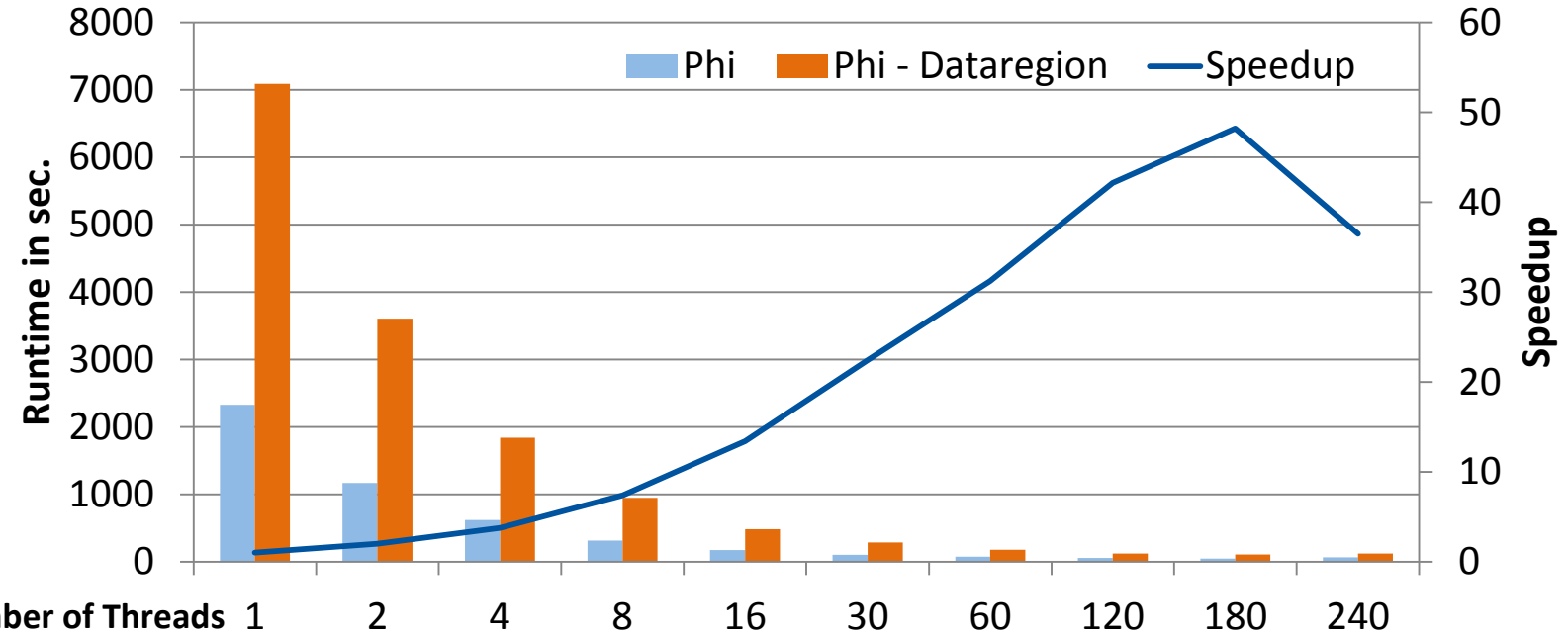
  SMXV:
  - 3.2 GB Transfer / SMXV Call
  - 3.3 s CPU-Time
  - 0.5 s Phi-Time
- Expand OpenMP Region - > Create Encompassing Target Data Region



Extended Target Region & Tuned Code

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Same Optimization as with the CPU Version**
- **Facilitated whole CG-Kernel on the Phi-Coprocessor**



SCIENTIFIC COMPUTING

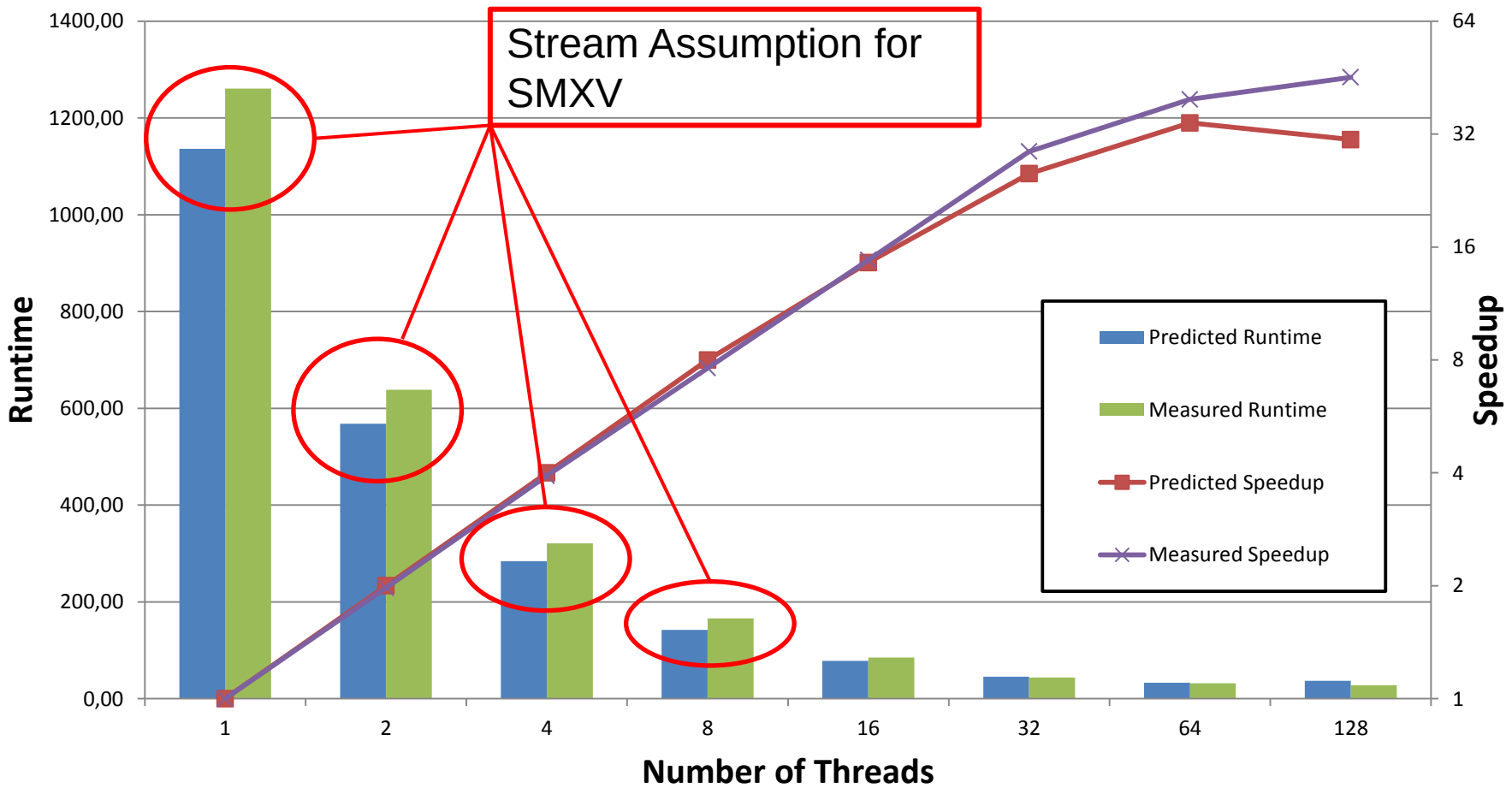
Result Verification

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Performance Model for an OpenMP CG Implementation**
  - Inputarguments:
    Number of Non-Zeros, Vector-Length, Number of Iterations, Number of Threads
  - SMXV: 14 Bytes / FP Operation
  - Vector-operations: 8 Bytes / FP Operation

  ⇒ Roofline: Memory Sub-System
  - Assume streaming performance (CPU + Memory System)
    Measurement of T_{FLOP}; T_{Byte}
  - 2 Sychronizationpoints (OpenMP reduction) necessary
    Measurement of $T_{Red.Overhead}(nT)$

Model vs Best Effort Implementation



Conclusion

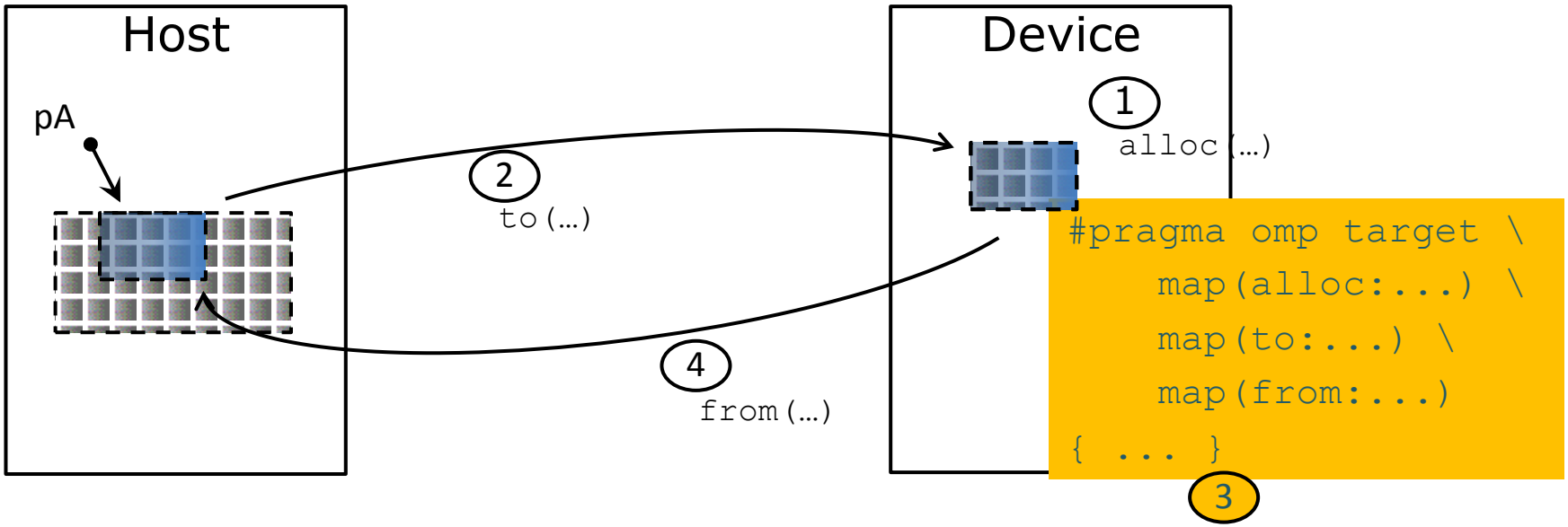
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Outlined Course Contents Necessary for OpenMP Implementation**
- **Workflow Suitable for CG Kernel**
- **Workflow also Applies to OpenMP 4.0 Accelerator Target-Style**
- **Performance Modeling can Indicate when to Stop**

Questions?

Execution + Data Model

- **Data environment is lexically scoped**
  - Data environment is destroyed at closing curly brace
  - Allocated buffers/data are automatically released
- **Use target construct to**
  - Transfer control from the host to the device
  - Establish a data environment (if not yet done)
  - Host thread waits until offloaded region completed



Example: Execution and Data Model

- **Environment Variable OMP_DEFAULT_DEVICE=<int>: sets the device number to use in target constructs**

```
double B[N] = ...; // some initialization
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for (i=0; i<N; i++)
   B[i] += sin(B[i]);
```

  - map variable B to device, then execute parallel region on device, works probably pretty well on Intel Xeon Phi

```
double B[N] = ...; // some initialization
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
for (i=0; i<N; i += num_blocks)
   #pragma omp parallel for
   for (b = i; b < i+num_blocks; b++)
      B[b] += sin(B[b]);
```

  - same as above, but code probably better optimized for NVIDIA GPGPUs

Comparing OpenMP with OpenACC

- **OpenMP 4.0 – for Intel Xeon Phi:**

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for (i=0; i<N; i++)
   B[i] += sin(B[i]);
```

- **OpenMP 4.0 – for NVIDIA GPGPU:**

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
for (i=0; i<N; i += num_blocks)
   #pragma omp parallel for
   for (b = i; b < i+num_blocks; b++)
      B[b] += sin(B[b]);
```

- **OpenACC – for NVIDIA GPGPU:**

```
#pragma acc parallel copy(B[0:N]) num_gangs(numblocks)\
            vector_length(bsize)
#pragma acc loop gang vector
  for (i=0; i<N; ++i) {
    B[i] += sin(B[i]);
  }
```

Comparing OpenMP with OpenACC

- **OpenMP 4.0 – for Intel Xeon Phi:**

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp parallel for
for (i=0; i<N; i++)
   B[i] += sin(B[i]);
```

- **OpenMP 4.0 – for NVIDIA GPGPU:**

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
for (i=0; i<N; i += num_blocks)
   #pragma omp parallel for
   for (b = i; b < i+num_blocks; b++)
      B[b] += sin(B[b]);
```

- **OpenACC – for NVIDIA GPGPU:**

Changed to RC2:

Combined directive

#pragma omp teams distribute parallel for

```
#pragma acc parallel copy(B[0:N]) num_gangs(numblocks)\
            vector_length(bsize)
#pragma acc loop gang vector
  for (i=0; i<N; ++i) {
    B[i] += sin(B[i]);
  }
```