# A Scalable Algebraic Multigrid Implementation for Non-Linear Elasticity Problems

Aurel Neic

Institute of Biomechanics

Medical University Graz

September 17, 2014

# Outline

AMG for elasticity problems

# Introduction

# Introduction

- Solve $K\,u = f$ using a hierarchy of equation systems

$$K^\ell\,\bar{u}^\ell = \bar{f}^\ell \quad ,\ell = 0,\ldots,L$$

- Hierarchy is constructed from *algebraic* operator $K$

$$K^{\ell+1} = R^\ell K^\ell P^\ell$$

# Exact Two-Grid Method

- Block System

$$\begin{bmatrix} K_{CC} & K_{CF} \\ K_{FC} & K_{FF} \end{bmatrix} \begin{bmatrix} u_C \\ u_F \end{bmatrix} = \begin{bmatrix} f_C \\ f_F \end{bmatrix}$$

- Schur decomposition

$$\begin{bmatrix} K_{CC} & K_{CF} \\ K_{FC} & K_{FF} \end{bmatrix} = \begin{bmatrix} I & K_{CF}K_{FF}^{-1} \\ & I \end{bmatrix} \begin{bmatrix} S_C & \\ & K_{FF} \end{bmatrix} \begin{bmatrix} I & \\ K_{FF}^{-1}K_{FC} & I \end{bmatrix}$$

$$S_C = K_{CC} - K_{CF}\,K_{FF}^{-1}\,K_{FC}$$

- Exact two grid method

$$P := \begin{pmatrix} I \\ -K_{FF}^{-1}K_{FC} \end{pmatrix}, \quad R := \begin{pmatrix} I & -K_{CF}K_{FF}^{-1} \end{pmatrix}, \quad G := \begin{pmatrix} 0 & 0 \\ 0 & K_{FF}^{-1} \end{pmatrix}$$

$$S_C = RKP, \quad M = (I - GK)(I - P\,S_C^{-1}\,R\,K) = 0$$

$$u_{k+1} := (I - M)K^{-1}f + Mu_k, \quad k \in \mathbb{N}$$

# Introduction

- Strong connection criterion

$$| K[i,j] | > \epsilon | K[i,i] |$$

- Construction of Prolongation operator

$$P := \begin{pmatrix} I_{CC} \\ P_{FC} \end{pmatrix}$$

$$\begin{aligned} n_i &:= | \{ j \in C : K[i,j] > \epsilon K[i,i] \} | \\ P_{FC}[i,j] &:= \begin{cases} 1/n_i & \text{if } | K[i,j] | > \epsilon | K[i,i] | \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

AMG for elasticity problems

# Adaptation

# Adaptation of AMG for Elasticity Problems

- Extension of the available AMG algorithm.
- Auxiliary matrix for coarsening based on $\|K_{i,j}^{3\times 3}\|$ matrix norms
- Extended prolongation operator based on auxiliary matirx
- Future adaptations: Vanka smoothers, K-cycle AMG

Parallel Scaling

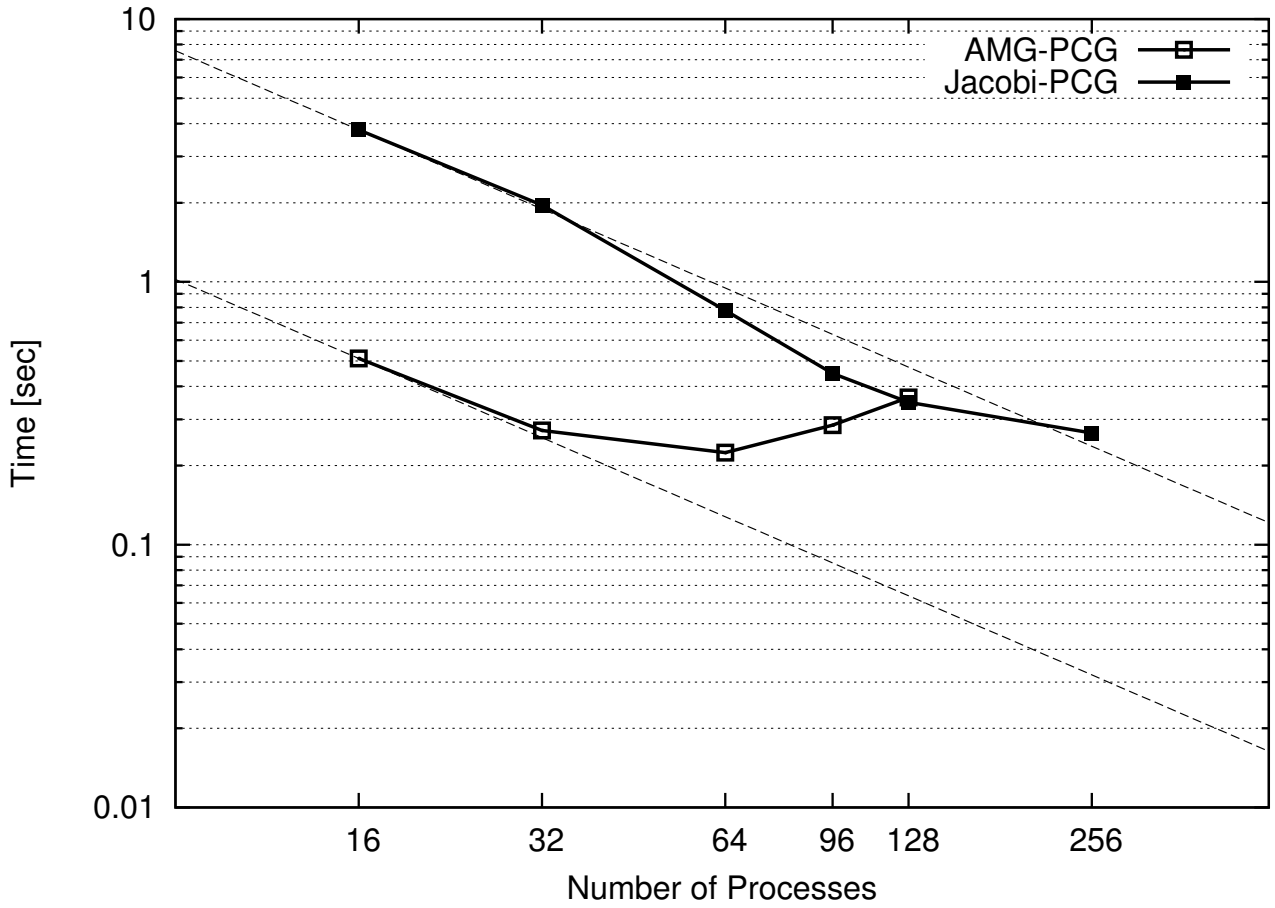
# Redistribution of AMG Hierarchy

# Introduction

- Multigrid methods efficient as solvers and preconditioners:
  Stable convergence w.r.t. grid refinement, spatial resolution
- Commonly used for elliptic PDEs.
- With the rise of multicore computing architectures:
  Concern about parallel scalability.
  Baker, Allison H. and Schulz, Martin and Yang, Ulrike M., *On the performance of an algebraic multigrid solver on multicore clusters*
- **Main bottleneck is communication on intermediate grids**

# Scaling Problems

Compare AMG-PCG with Jacobi-PCG as a limit case.



# Scaling Problems

- Fine-Grid
  - Big subdomains, interfaces, message sizes
  - Sparse communication pattern
  - $\rightarrow$ Point-To-Point type
- Coarse-Grid
  - Small subdomains, interfaces, message sizes
  - Few processes, accumulate interface values
  - $\rightarrow$ Scatter/Gather type
- Intermediate Grids
  - Small subdomains, interfaces, message sizes
  - Still most processes, accumulate interface values
  - $\rightarrow$ All-To-All type

# Redistribution of MG hierarchy

- Common solution: Reduce parallelism
  - Redistribute parallel data on fewer processes
- Open questions:
  - When to redistribute: Once, multiple times, systematic?
  - Redistribution pattern: Based an which criteria?

# Proposed Method

- Systematic reduction of processes, rate is configurable.
- Redistribution in block-pattern.

**Reasoning:**

- Systematic:
  - Logarithmic reduction of processes, reduction of unknowns should be much higher
  - Robust configuration, doesn't need fine tuning.
- Block-Pattern:
  - Should match hardware layout
  - Intermediate levels redistributed in local memory.

# Block Pattern

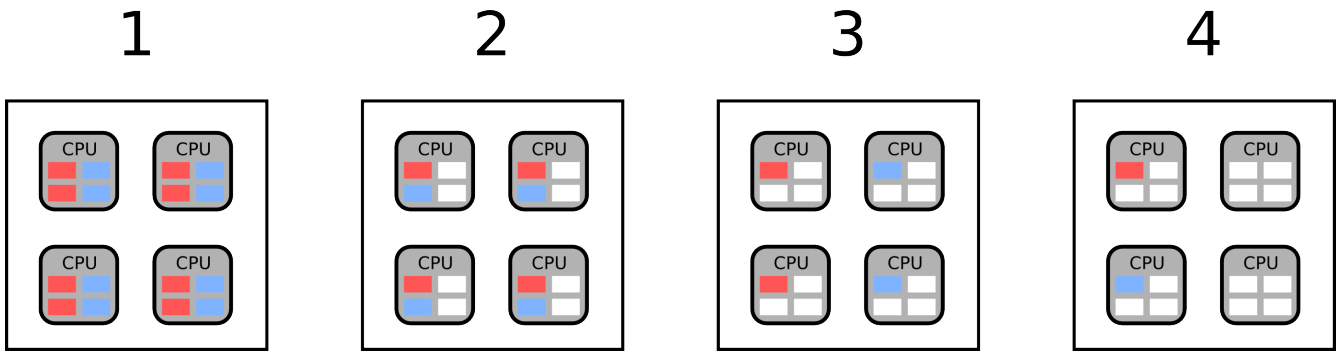
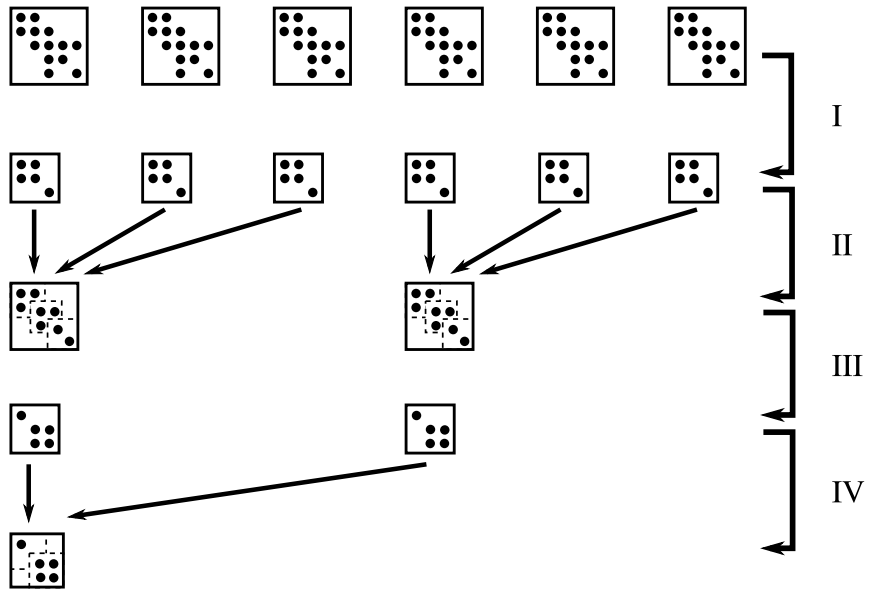



Figure: Redistribution pattern for a blocksize of 2. **Red:** Active Ranks, **Blue:** Idle Ranks

- Can be computed based on process rank if the assignment to computing units was continuous.
- In case of 16 processes per compute node, the first 4 redistributions are in shared memory!

# AMG Setup

1. Coarsening
   - 1.1 Coarse-grid selection
   - 1.2 Generation of Prolongation/Restriction
   - 1.3 Triple-matrix-product
2. Redistribution
   - 2.1 Active-idle selection
   - 2.2 Data transfer
   - 2.3 Matrix accumulation



# Adapted V-Cycle

**Require:** $\mathsf{K}_p^\ell, \underline{\mathsf{f}}_p^\ell, \underline{\mathsf{u}}_p^\ell, \underline{\mathsf{r}}_p^\ell, P_p^\ell, J_p^\ell, G, a \in G, A_p, active_p; \ell = 1, \ldots, L$

**if** $\ell < L$ **then**

**if** $active_p[\ell] = true$ **then**

$\underline{\mathsf{u}}_p^\ell \leftarrow 0$

$\underline{\mathsf{u}}_p^\ell \leftarrow J_p^\ell(\underline{\mathsf{u}}_p^\ell, \underline{\mathsf{f}}_p^\ell)$ {Pre-smooth}

$\underline{\mathsf{r}}_p^\ell \leftarrow \underline{\mathsf{f}}_p^\ell - \mathsf{K}_p^\ell \underline{\mathsf{u}}_p^\ell$ {Compute residual}

$\underline{\mathsf{f}}_p^{\ell+1} \leftarrow (P_p^\ell)^T \underline{\mathsf{r}}_p^\ell$ {Restrict residual}

$\underline{\mathsf{f}}_a^{\ell+1} \leftarrow \sum_{i \in G} \hat{A}_a A_i^T \underline{\mathsf{f}}_i^{\ell+1}$ {Gather}

**end if**

multigrid$(\ell + 1)$ {Recursion}

**if** $active_p[\ell] = true$ **then**

$\underline{\mathsf{u}}_i^{\ell+1} \leftarrow A_i \hat{A}_a^T \underline{\mathsf{u}}_a^{\ell+1} \quad i \in G$ {Scatter}

$\underline{\mathsf{c}}_p^\ell \leftarrow P_p^\ell \underline{\mathsf{u}}_p^{\ell+1}$ {Prolongate correction}

$\underline{\mathsf{u}}_p^\ell \leftarrow \underline{\mathsf{u}}_p^\ell + \underline{\mathsf{c}}_p^\ell$ {Add correction}

$\underline{\mathsf{u}}_p^\ell \leftarrow J_p^\ell(\underline{\mathsf{u}}_p^\ell, \underline{\mathsf{f}}_p^\ell)$ {Post-smooth}

**end if**

**else**

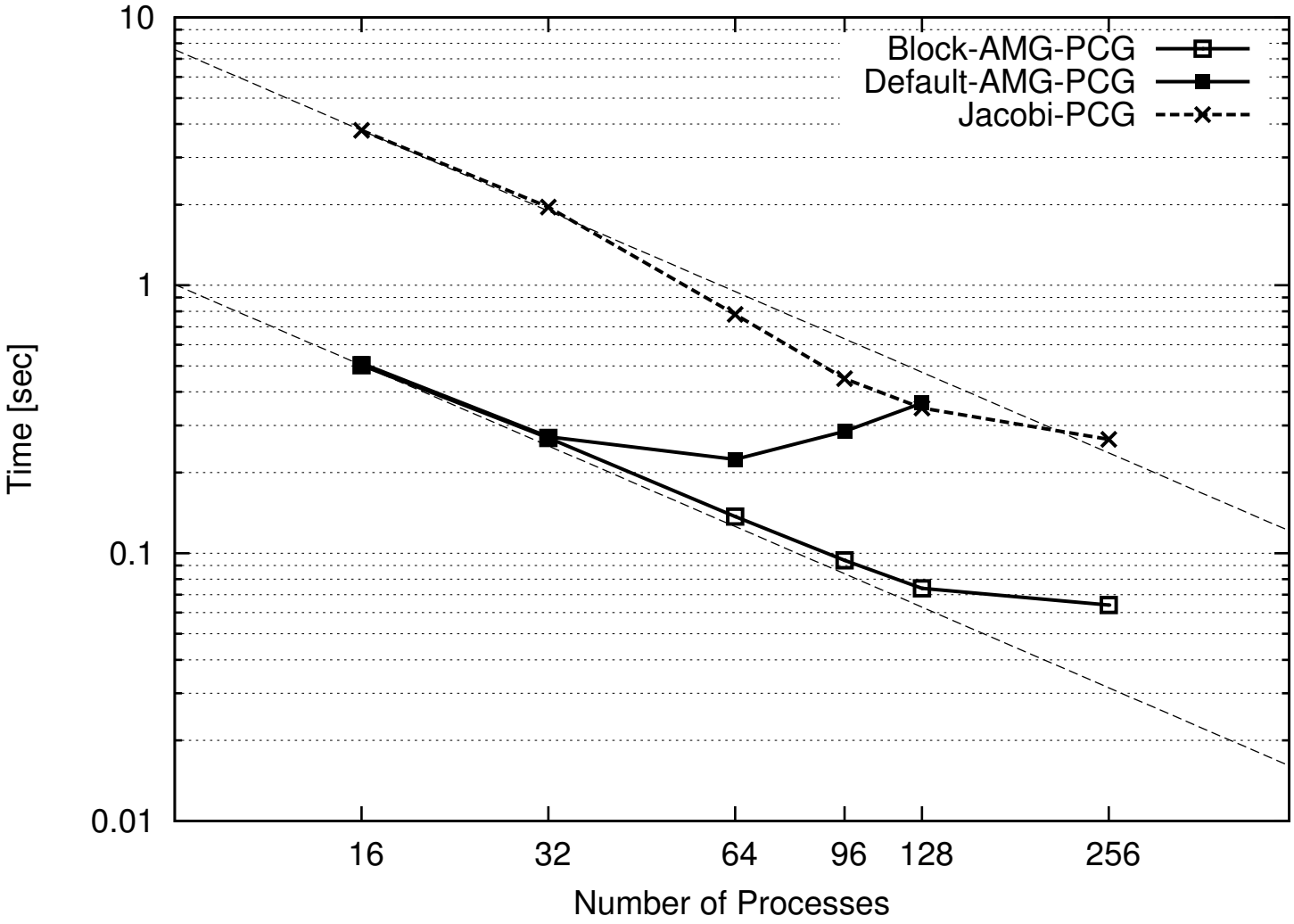
**if** $active_p[L] = true$ **then**

$\underline{\mathsf{u}}_p^L \leftarrow (\mathsf{K}_p^L)^{-1} \underline{\mathsf{f}}_p^L$ {Direct solve}

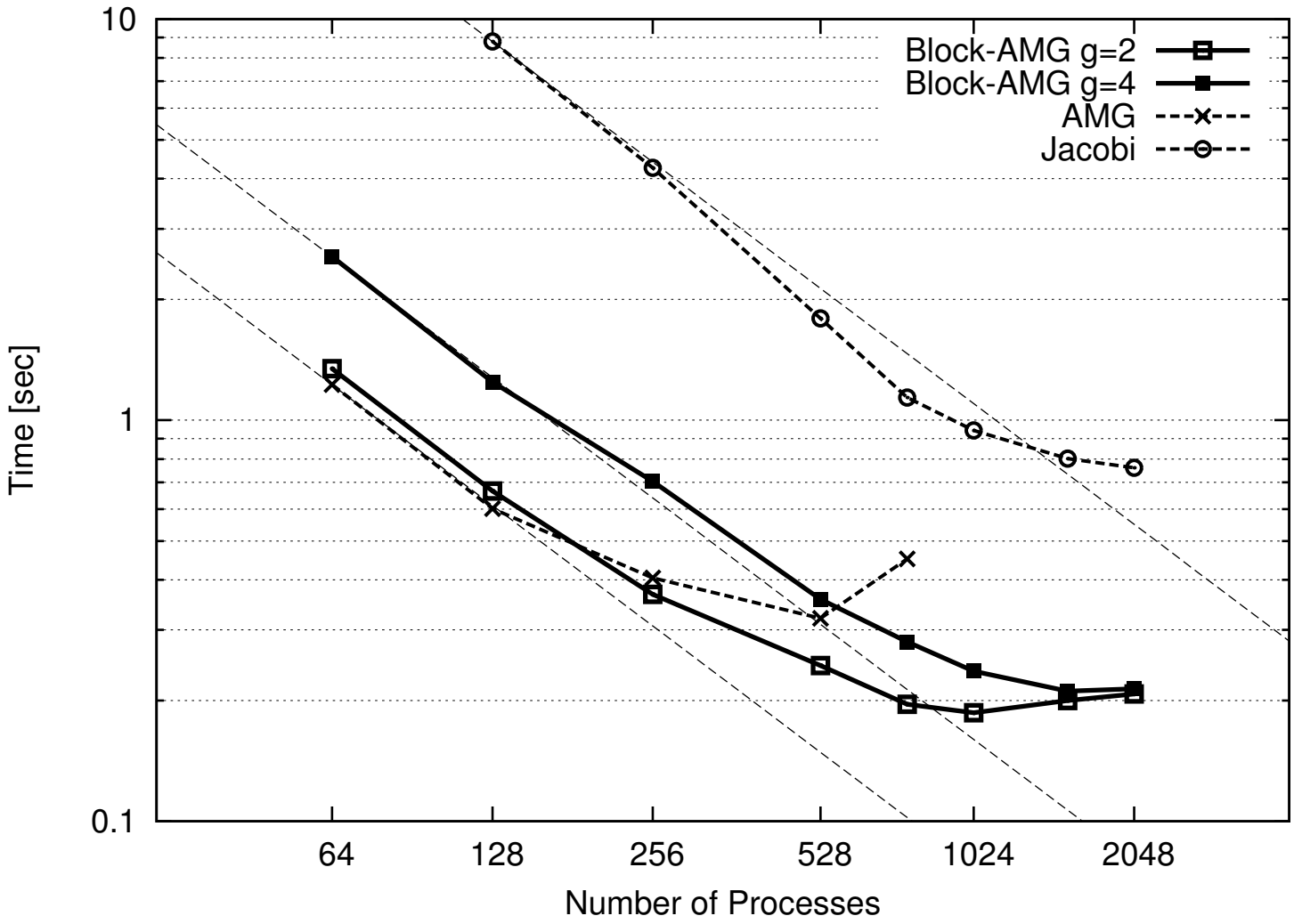
**end if**

**end if**

# Benchmarks


Block-AMG-PCG
Default-AMG-PCG
Jacobi-PCG
Time [sec]
10
1
0.1
0.01
16
32
64
96
128
256
Number of Processes

# Benchmarks


10
1
0.1
Time [sec]
64
128
256
528
1024
2048
Number of Processes
Block-AMG g=2
Block-AMG g=4
AMG
Jacobi

Parallel Scaling

# Hybrid MPI/OpenMP Parallelization

# Motivation

- Increase the number of OpenMP threads per MPI process
  - SuperMuc Thin Node: 16 cores, 32 GB RAM
  - SuperMuc Fat Node: 40 core, 256 GB RAM
- Pin OpenMP threads to computing tasks
  - Desktop: OpenMPI rankfiles
  - Server: OpenMPI rankfiles, Custom modules
- Advantage: More computing tasks with less communication

# Motivation

**Matrix-Vector benchmarks:**

| | runtime | | speedup | |
|---|---|---|---|---|
| # cores | OMP | MPI | OMP | MPI |
| 1 | 4.579 | 4.336 | 1.00 | 1.00 |
| 2 | 2.306 | 2.161 | 1.99 | 2.01 |
| 4 | 1.347 | 1.123 | 3.40 | 3.86 |
| 6 | 1.133 | 1.009 | 4.04 | 4.30 |
| 12 | 0.565 | 0.378 | 8.10 | 11.48 |

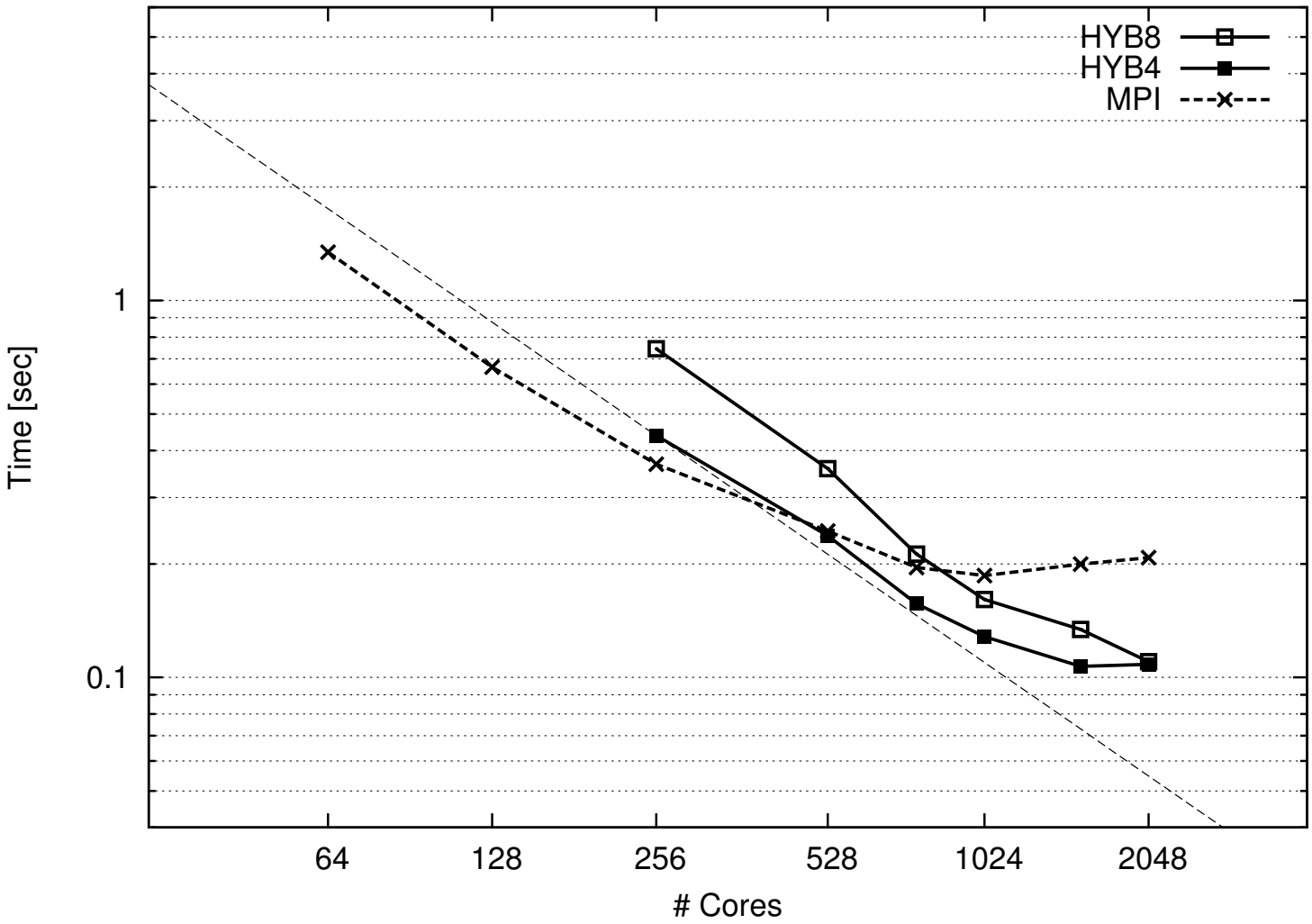
**OpenMP kernels show reduced computational efficiency**

# Implementation

- OpenMP parallelization only for computational kernels
- Independent of MPI parallelization
- Scheduler of choice: **guided**
  - Background load should be assumed $\rightarrow$ dynamic scheduling usually better
  - Optimal chunksize unknown or varying $\rightarrow$ guided scheduling decreases the chunksize linearly to a defined minimum size
- Optimal number of threads per process is hardware and problem dependent
- Parallelization across multiple CPUs disadvantageous

# Benchmarks

Reduced number of MPI processes is advantageous for a large number of threads.



Thank you!

The research was funded by the Austrian Science Fund (FWF)