

Algebraic Multigrid Methods on GPU-Accelerated Hybrid Architectures

Manfred Liebmann
Technische Universität München
Chair of Optimal Control
Center for Mathematical Sciences, M17
`manfred.liebmann@tum.de`



Technische Universität München



Fakultät für Mathematik

October 20, 2015

(8) Bidomain Equations with Elasticity Coupling

The elastic coupling is realized with the parameters \mathbf{a} and the Cauchy-Green deformation tensor \mathbf{C} , calculated from the deformation model [Pathmanathan/Whiteley 2009], with the equilibrium equations of large elastic deformations describing the mechanic behavior.

$$-\nabla \cdot \boldsymbol{\sigma}_e \mathbf{C}^{-1} \nabla \phi_e = \nabla \cdot \boldsymbol{\sigma}_i \mathbf{C}^{-1} \nabla \phi_i + I_e \quad (1)$$

$$\beta I_m = \nabla \cdot \boldsymbol{\sigma}_i \mathbf{C}^{-1} \nabla \phi_i \quad (2)$$

$$I_m = C_m \frac{\partial V_m}{\partial t} + I_{ion}(V_m, \boldsymbol{\eta}, \mathbf{a}) - I_i \quad (3)$$

$$\frac{d\boldsymbol{\eta}}{dt} = f(t, \boldsymbol{\eta}, \mathbf{a}) \quad (4)$$

$$V_m = \phi_i - \phi_e \quad (5)$$

Large Elastic Deformations

$$-\operatorname{div} \boldsymbol{\sigma}(u) = f, \quad \boldsymbol{\sigma} = 2J^{-1} \mathbf{F} \frac{\partial \Psi(\mathbf{C})}{\partial \mathbf{C}} \mathbf{F}^\top \quad (6)$$

$$\mathbf{C} = \mathbf{F}^\top \mathbf{F}, \quad J = \det \mathbf{F}, \quad \mathbf{F} = \mathbf{F}(u) \quad (7)$$

$$(8)$$

Where

$$\Psi = U(J) + \bar{\Psi}_{\text{iso}} + \bar{\Psi}_{\text{aniso}} + \bar{\Psi}_{\text{act}}(\mathbf{a}, \eta(V_m)), \quad U(J) = \frac{\kappa}{2}(J - 1)^2. \quad (9)$$

is the Helmholtz free energy including both the isotropic and anisotropic response, and an active part due to the excitation-induced contraction during the course of depolarization. [Kroon/Holzapfel 2009; Pathmanathan/Whiteley 2009]

Nonlinear Elasticity: A Sketch

Solve the nonlinear system of equations

$$K(u) \cdot u = f$$

with u describing the elastic deformation with Newtons method.

In every Newton iteration we have to solve the system

$$A'(u) \cdot \Delta u = g(u) \tag{10}$$

where

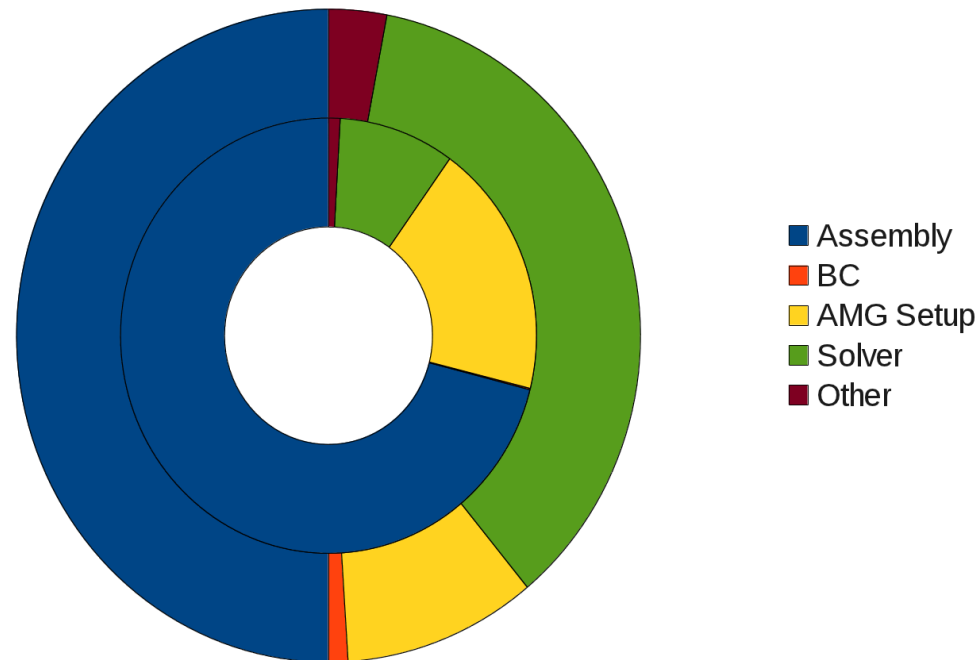
$$\begin{aligned} A'(u) &= \frac{\partial K(u)}{\partial u} \cdot u + K(u) \\ g(u) &= f - K(u) \cdot u \end{aligned} \tag{11}$$

Nonlinear Elasticity: Algorithmic View I

Newton solver for the nonlinear elasticity problem:

1. Calculate $K(u)$, $A'(u)$, $g(u)$
2. Solve the Newton system with AMG-PCG
 - (a) Setup of AMG-preconditioner wrt. $A'(u)$
 - i. Find coarse/fine nodes
 - ii. Calculate interpolation operators
 - iii. Calculate coarse grid operators
 - (b) Apply AMG-PCG solver
3. Update $\hat{u} = u + \Delta u$
4. Goto 1

CARP-PT Performance of Elasticity Solver



	CPU %	GPU %
Assembly	50	71
AMG Setup	10	19
AMG-PCG Solver	36	9

Table 1: Percentage of time spent in the components of the nonlinear elasticity solver

Nonlinear Elasticity: Algorithmic View II

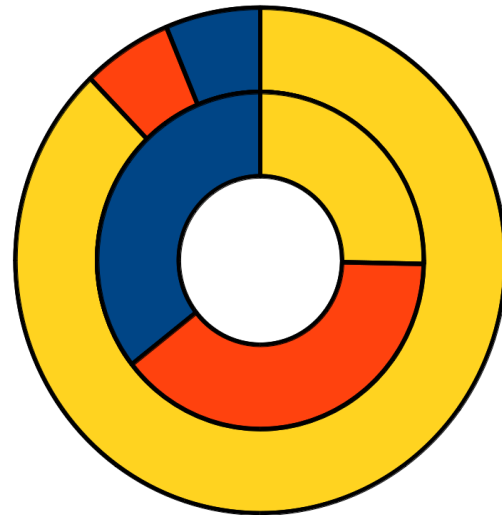
1. Setup of operators performed once
 - (a) Calculate sparsity pattern of $K(u)$, $A'(u)$
 - (b) Calculate $K(u)$, $A'(u)$, $g(u)$
 - (c) Setup of AMG-preconditioner wrt. $A'(u)$

2. Solve the Newton system with AMG-PCG
 - (a) Setup of AMG-preconditioner wrt. $A'(u)$ with fixed sparsity pattern
 - i. No setup wrt. coarse/fine nodes
 - ii. Update interpolation operators
 - iii. Update coarse grid operators
 - (b) Apply AMG-PCG solver

3. Update $\hat{u} = u + \Delta u$
Update $K(\hat{u})$, $A'(\hat{u})$, $g(\hat{u})$ with fixed sparsity pattern

4. Goto 2

CARP-PT Performance of Optimized Elasticity Solver



■ Assembly ■ AMG setup ■ Solver

	Original		Optimized	
	CPU %	GPU %	CPU %	GPU %
Assembly	50	71	6	36
AMG Setup	10	19	6	39
PCG-AMG Solver	36	9	88	25

Table 2: Percentage of time spent in the components of the nonlinear elasticity solver

(9) AMG for Elasticity Revisited

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

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

- 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

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

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

- Exact two grid method

$$R := \left(I \quad -K_{CF}K_{FF}^{-1} \right), \quad P := \begin{pmatrix} I \\ -K_{FF}^{-1}K_{FC} \end{pmatrix}$$

Standard AMG Coarsening

- Strong connection criterion

$$|K_{ij}| > \epsilon |K_{ii}|$$

- Construction of Prolongation operator

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

$$\begin{aligned} n_i &:= \#\{j \in C : |K_{ij}| > \epsilon |K_{ii}|\} \\ (P_{FC})_{ij} &:= \begin{cases} 1/n_i & \text{if } |K_{ij}| > \epsilon |K_{ii}| \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Adaptation of AMG for Elasticity Problems

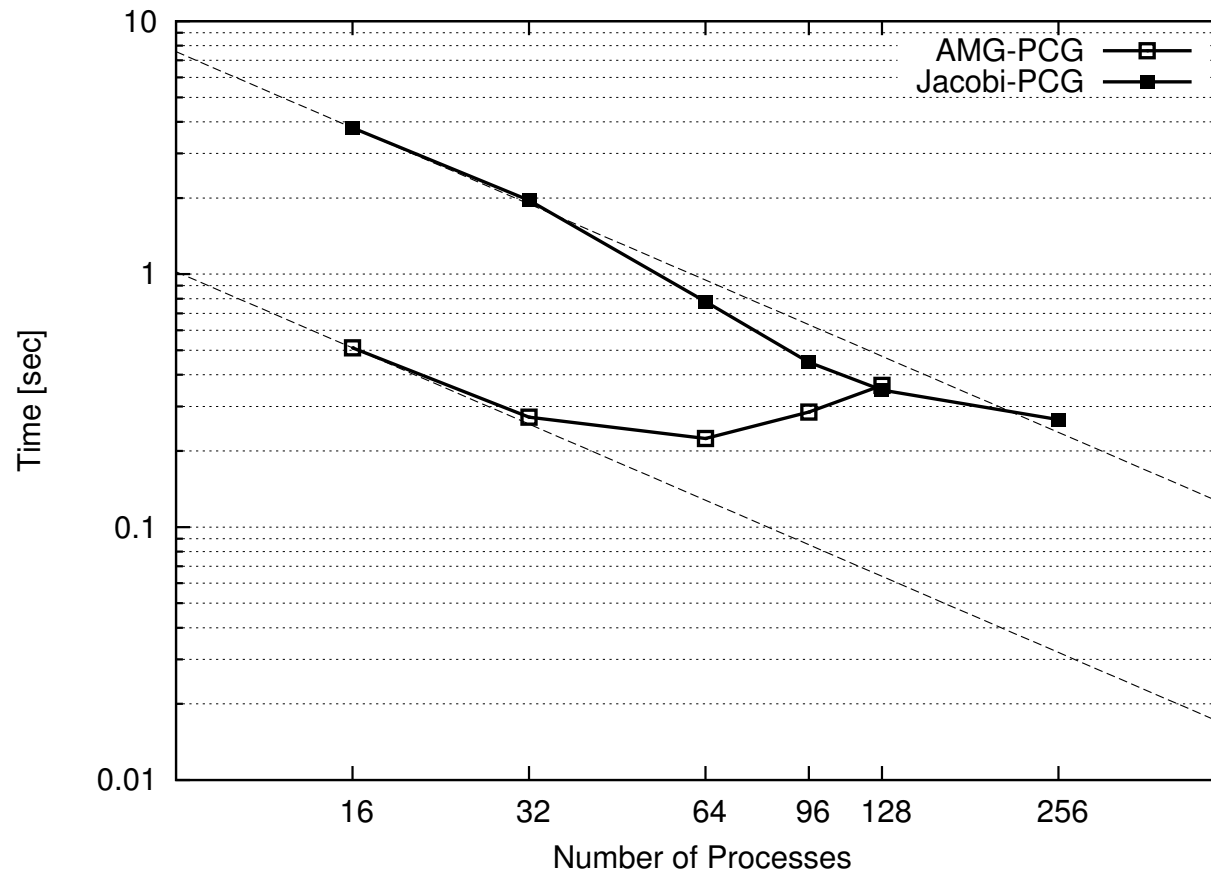
- Extension of the standard AMG algorithm
- The matrix K has a 3×3 block structure
- Define an auxiliary matrix \tilde{K} by taking matrix norms of the 3×3 blocks of K
- Calculate coarsening with auxiliary matrix \tilde{K}
- Extended prolongation operator based on auxiliary matrix \tilde{K}

(10) Parallel Scaling: Redistribution of AMG Hierarchy

- Multigrid methods efficient as solvers and preconditioners:
Stable convergence w.r.t. grid refinement and spatial resolution
- Main concern with modern hardware architectures: **Parallel scalability**
- **Main bottleneck is communication on intermediate grids**

Scaling Problems

Comparison of AMG-PCG and Jacobi-PCG as a limit case.



Scaling Problems

- Fine-Grid
 - Big subdomains, interfaces, message sizes
 - Sparse communication pattern
- Intermediate Grids
 - Small subdomains, interfaces, message sizes
 - Still most processes, accumulate interface values
- Coarse-Grid
 - Small subdomains, interfaces, message sizes
 - Few processes, accumulate interface values

Redistribution of multigrid hierarchy

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

Proposed Method

- Systematic reduction of processes with configurable rate
- Redistribution in a block-pattern matching the domain decomposition

Reasoning:

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

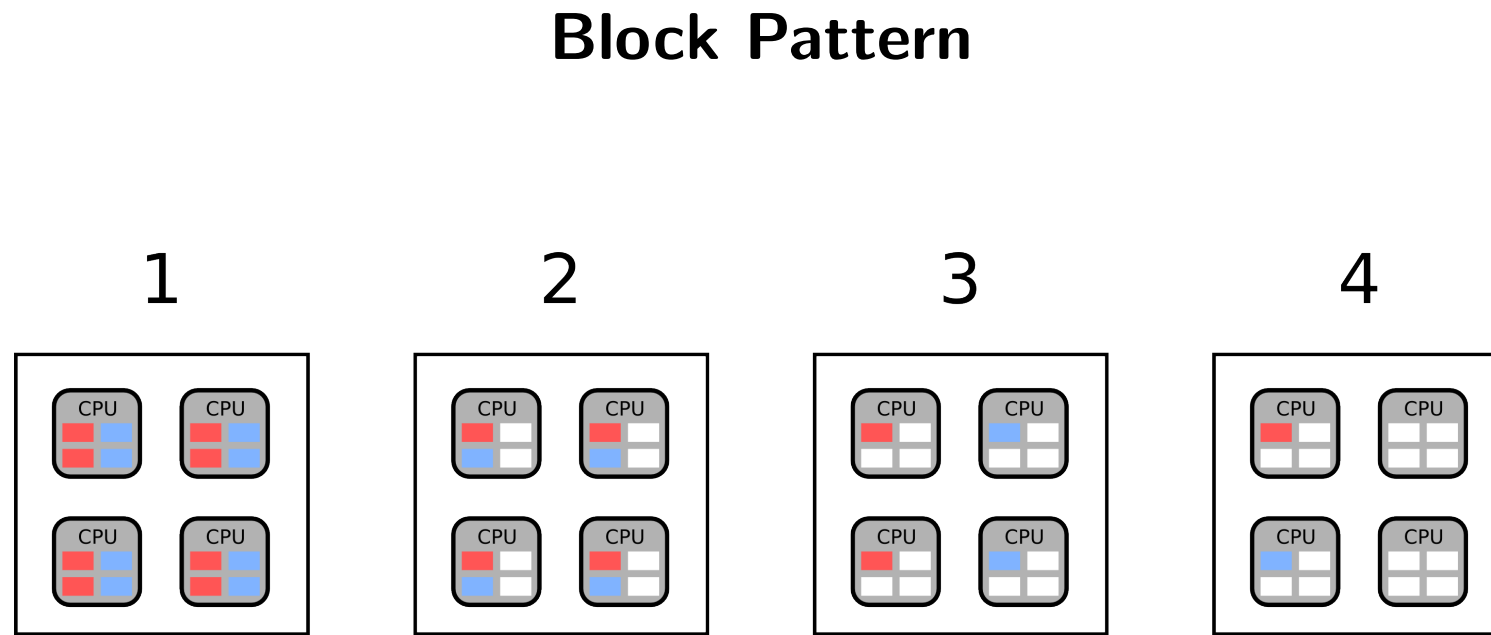


Figure 1: Redistribution pattern for a blocksize 2. **Red:** Active Ranks, **Blue:** Idle Ranks

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

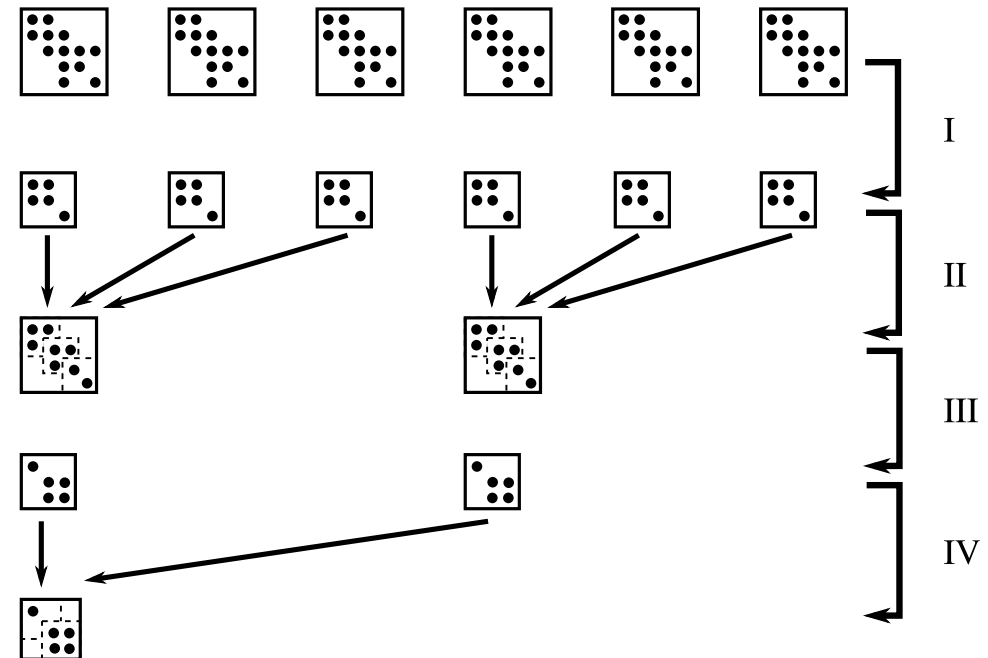
Modified AMG Setup and Algorithm

1. Coarsening

- (a) Coarse-grid selection
- (b) Construct Prolongation/Restriction
- (c) Triple-matrix-product

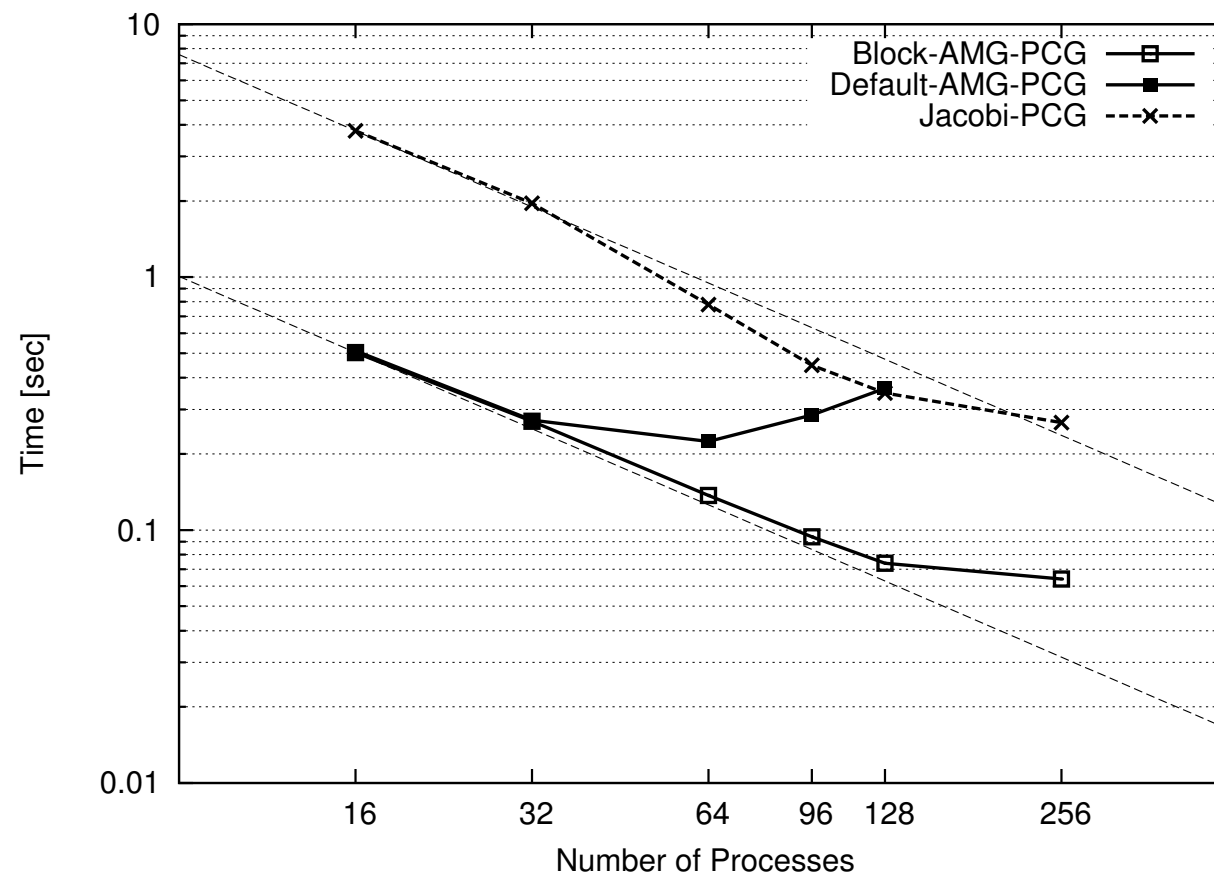
2. Redistribution

- (a) Active-idle selection
- (b) Data transfer
- (c) Matrix accumulation

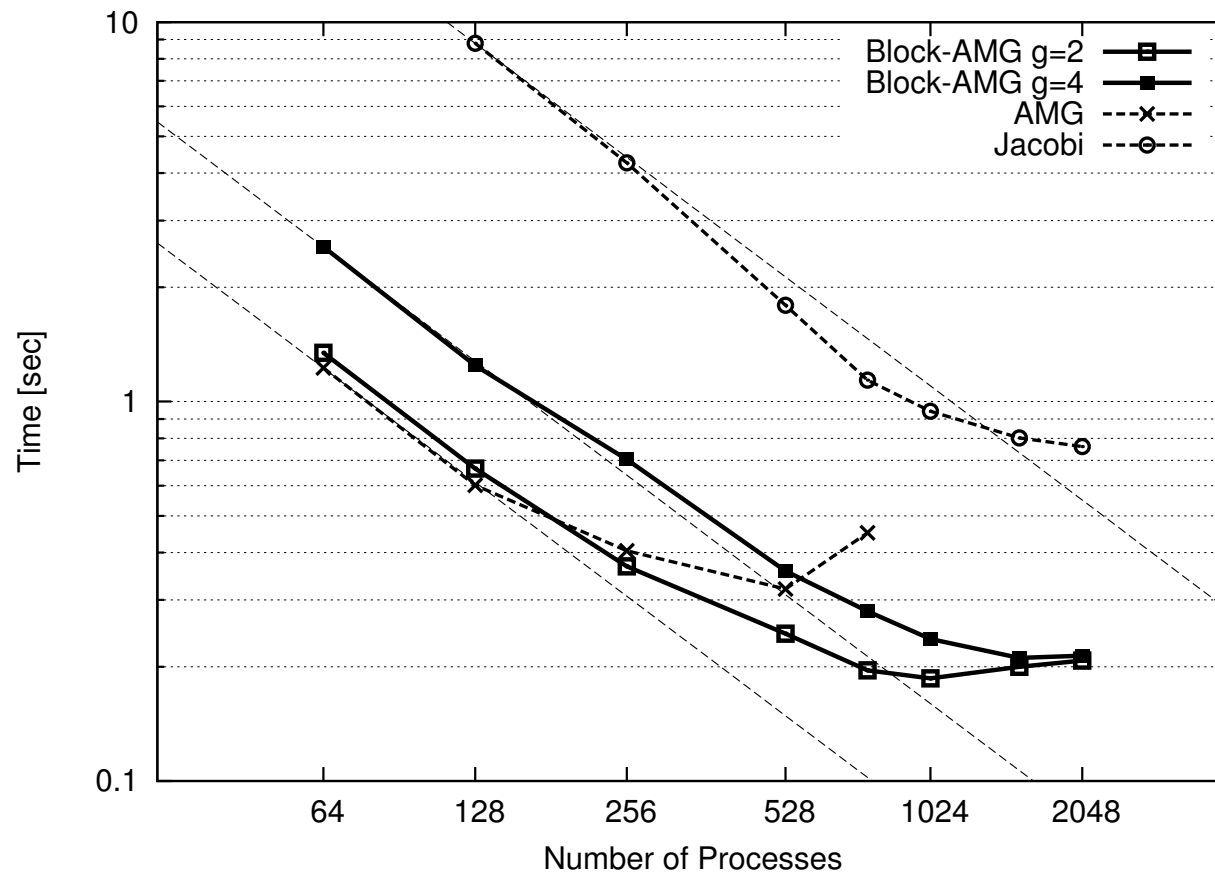


Require: $\mathbf{K}_p^\ell, \underline{\mathbf{f}}_p^\ell, \underline{\mathbf{u}}_p^\ell, \underline{\mathbf{r}}_p^\ell, P_p^\ell, J_p^\ell, G, a \in G, A_p, active_p; \ell = 1, \dots, L$
if $\ell < L$ **then**
 if $active_p[\ell] = true$ **then**
 $\underline{\mathbf{u}}_p^\ell \leftarrow 0$
 $\underline{\mathbf{u}}_p^\ell \leftarrow J_p^\ell(\underline{\mathbf{u}}_p^\ell, \underline{\mathbf{f}}_p^\ell)$ {Pre-smooth}
 $\underline{\mathbf{r}}_p^\ell \leftarrow \underline{\mathbf{f}}_p^\ell - \mathbf{K}_p^\ell \underline{\mathbf{u}}_p^\ell$ {Compute residual}
 $\underline{\mathbf{f}}_p^{\ell+1} \leftarrow (P_p^\ell)^T \underline{\mathbf{r}}_p^\ell$ {Restrict residual}
 $\underline{\mathbf{f}}_a^{\ell+1} \leftarrow \sum_{i \in G} \hat{A}_a A_i^T \underline{\mathbf{f}}_i^{\ell+1}$ {Gather}
 end if
 multigrid($\ell + 1$) {Recursion}
 if $active_p[\ell] = true$ **then**
 $\underline{\mathbf{u}}_i^{\ell+1} \leftarrow A_i \hat{A}_a^T \underline{\mathbf{u}}_a^{\ell+1} \quad i \in G$ {Scatter}
 $\underline{\mathbf{c}}_p^\ell \leftarrow P_p^\ell \underline{\mathbf{u}}_p^{\ell+1}$ {Prolongate correction}
 $\underline{\mathbf{u}}_p^\ell \leftarrow \underline{\mathbf{u}}_p^\ell + \underline{\mathbf{c}}_p^\ell$ {Add correction}
 $\underline{\mathbf{u}}_p^\ell \leftarrow J_p^\ell(\underline{\mathbf{u}}_p^\ell, \underline{\mathbf{f}}_p^\ell)$ {Post-smooth}
 end if
else
 if $active_p[L] = true$ **then**
 $\underline{\mathbf{u}}_p^L \leftarrow (\mathbf{K}_p^L)^{-1} \underline{\mathbf{f}}_p^L$ {Direct solve}
 end if
end if

Benchmarks 1



Benchmarks 2



Hybrid MPI/OpenMP Parallelization

- 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:

# cores	runtime		speedup	
	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 → dynamic scheduling usually better
 - Optimal chunksize unknown or varying → 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.

