Parallel implementation of discontinuous Galerkin method for compressible flow simulations

Michal Zajac, Vít Dolejší

Charles University Prague
Faculty of Mathematics and Physics

Programy a algoritmy numerické matematiky 14,
Dolní Maxov, Czech Republic,
June 01-06, 2008

Outline

1 Introduction

2 Parallel implementation

3 Outlook

Outline

1. Introduction
2. Parallel implementation
3. Outlook

Outline

1. Introduction
2. Parallel implementation
3. Outlook

Introduction

- **Our aim**: efficient numerical scheme for the solution of the compressible Navier-Stokes equations,

$$\frac{\partial \mathbf{w}}{\partial t} = \nabla \cdot \mathbf{G}(\mathbf{w}, \nabla \mathbf{w}), \qquad \mathbf{w} : \Omega \times (0,T) \to R^4, \qquad (1)$$

- space semi-discretization: $\mathbf{w}(x,t) \approx \mathbf{w}_h(t) \in \mathbf{S}_h, \; t \in (0,T)$

$$\frac{\partial}{\partial t}\mathbf{w}_h = F(t, \mathbf{w}_h, \nabla \mathbf{w}_h), \qquad \mathbf{w}_h : (0,T) \to \mathbf{S}_h, \qquad (2)$$

Introduction

- **Our aim**: efficient numerical scheme for the solution of the compressible Navier-Stokes equations,

$$\frac{\partial \mathbf{w}}{\partial t} = \nabla \cdot \mathbf{G}(\mathbf{w}, \nabla \mathbf{w}), \qquad \mathbf{w} : \Omega \times (0, T) \to R^4, \qquad (1)$$

- space semi-discretization: $\mathbf{w}(x, t) \approx \mathbf{w}_h(t) \in \mathbf{S}_h, \; t \in (0, T)$

$$\frac{\partial}{\partial t}\mathbf{w}_h = F(t, \mathbf{w}_h, \nabla \mathbf{w}_h), \qquad \mathbf{w}_h : (0, T) \to \mathbf{S}_h, \qquad (2)$$

Introduction

- **Our aim**: efficient numerical scheme for the solution of the compressible Navier-Stokes equations,

$$\frac{\partial \mathbf{w}}{\partial t} = \nabla \cdot \mathbf{G}(\mathbf{w}, \nabla \mathbf{w}), \qquad \mathbf{w} : \Omega \times (0, T) \to \boldsymbol{R}^4, \qquad (1)$$

- space semi-discretization: $\mathbf{w}(x, t) \approx \mathbf{w}_h(t) \in \mathbf{S}_h,\ t \in (0, T)$

$$\frac{\partial}{\partial t}\mathbf{w}_h = F(t, \mathbf{w}_h, \nabla \mathbf{w}_h), \qquad \mathbf{w}_h : (0, T) \to \mathbf{S}_h, \qquad (2)$$

Introduction

- **Our aim**: efficient numerical scheme for the solution of the compressible Navier-Stokes equations,

$$\frac{\partial \mathbf{w}}{\partial t} = \nabla \cdot \mathbf{G}(\mathbf{w}, \nabla \mathbf{w}), \qquad \mathbf{w} : \Omega \times (0, T) \to \boldsymbol{R}^4, \qquad (1)$$

- space semi-discretization: $\mathbf{w}(x, t) \approx \mathbf{w}_h(t) \in \mathbf{S}_h, \; t \in (0, T)$

$$\frac{\partial}{\partial t}\mathbf{w}_h = F(t, \mathbf{w}_h, \nabla \mathbf{w}_h), \qquad \mathbf{w}_h : (0, T) \to \mathbf{S}_h, \qquad (2)$$

Introduction (2)

- full time-space discretization,

$$\left(\mathbf{M} + \tau_k \mathbf{C}(\mathbf{w}_h^k)\right) \mathbf{w}_h^{k+1} = \mathbf{F}(\mathbf{w}_h^k), \qquad (3)$$

  - $\mathbf{w}_h^k \in R^{\mathrm{dof}}, \; k = 1, 2, \ldots,$
  - $\mathbf{M}$ – mass matrix,
  - $\mathbf{C}$ – "discrete flux" matrix,
  - $\mathbf{F}$ – RHS (boundary conditions),
  - τ_k – time step,

Introduction (2)

- full time-space discretization,

$$\left(\mathbf{M} + \tau_k \mathbf{C}(\mathbf{w}_h^k)\right) \mathbf{w}_h^{k+1} = \mathbf{F}(\mathbf{w}_h^k), \qquad (3)$$

- $\mathbf{w}_h^k \in R^{\text{dof}}, \ k = 1, 2, \dots,$
- $\mathbf{M}$ – mass matrix,
- $\mathbf{C}$ – "discrete flux" matrix,
- $\mathbf{F}$ – RHS (boundary conditions),
- τ_k – time step,

Introduction (2)

- full time-space discretization,

$$\left(\mathbf{M} + \tau_k \mathbf{C}(\mathbf{w}_h^k)\right) \mathbf{w}_h^{k+1} = \mathbf{F}(\mathbf{w}_h^k), \qquad (3)$$

- $\mathbf{w}_h^k \in R^{\mathrm{dof}}, \ k = 1, 2, \dots,$
- $\mathbf{M}$ – mass matrix,
- $\mathbf{C}$ – "discrete flux" matrix,
- $\mathbf{F}$ – RHS (boundary conditions),
- τ_k – time step,

Introduction (2)

- full time-space discretization,

$$\left(\mathbf{M} + \tau_k \mathbf{C}(\mathbf{w}_h^k)\right) \mathbf{w}_h^{k+1} = \mathbf{F}(\mathbf{w}_h^k), \qquad (3)$$

  - $\mathbf{w}_h^k \in \boldsymbol{R}^{\mathrm{dof}}, \; k = 1, 2, \ldots,$
  - $\mathbf{M}$ – mass matrix,
  - $\mathbf{C}$ – "discrete flux" matrix,
  - $\mathbf{F}$ – RHS (boundary conditions),
  - τ_k – time step,

Introduction (2)

- full time-space discretization,

$$\left(\mathbf{M} + \tau_k \mathbf{C}(\mathbf{w}_h^k)\right) \mathbf{w}_h^{k+1} = \mathbf{F}(\mathbf{w}_h^k), \qquad (3)$$

  - $\mathbf{w}_h^k \in \boldsymbol{R}^{\mathrm{dof}},\ k = 1, 2, \ldots,$
  - **M** – mass matrix,
  - **C** – "discrete flux" matrix,
  - **F** – RHS (boundary conditions),
  - τ_k – time step,

Introduction (2)

- full time-space discretization,

$$\left(\mathbf{M} + \tau_k \mathbf{C}(\mathbf{w}_h^k)\right) \mathbf{w}_h^{k+1} = \mathbf{F}(\mathbf{w}_h^k), \quad (3)$$

  - $\mathbf{w}_h^k \in \boldsymbol{R}^{\mathrm{dof}}, \ k = 1, 2, \ldots,$
  - **M** – mass matrix,
  - **C** – "discrete flux" matrix,
  - **F** – RHS (boundary conditions),
  - τ_k – time step,

Introduction (2)

- full time-space discretization,

$$\left(\mathbf{M} + \tau_k \mathbf{C}(\mathbf{w}_h^k)\right) \mathbf{w}_h^{k+1} = \mathbf{F}(\mathbf{w}_h^k), \tag{3}$$

  - $\mathbf{w}_h^k \in \boldsymbol{R}^{\mathrm{dof}}, \ k = 1, 2, \dots,$
  - $\mathbf{M}$ – mass matrix,
  - $\mathbf{C}$ – "discrete flux" matrix,
  - $\mathbf{F}$ – RHS (boundary conditions),
  - τ_k – time step,

Introduction (2)

- full time-space discretization,

$$\left(\mathbf{M} + \tau_k \mathbf{C}(\mathbf{w}_h^k)\right) \mathbf{w}_h^{k+1} = \mathbf{F}(\mathbf{w}_h^k), \qquad (3)$$

  - $\mathbf{w}_h^k \in \boldsymbol{R}^{\text{dof}},\; k = 1, 2, \ldots,$
  - $\mathbf{M}$ – mass matrix,
  - $\mathbf{C}$ – "discrete flux" matrix,
  - $\mathbf{F}$ – RHS (boundary conditions),
  - τ_k – time step,

Linear algebra problem

- iterative solver necessary for industrial applications
  $\Rightarrow$ restarted GMRES with block diagonal preconditioning
- L^2-orthogonal basis of $\mathbf{S}_h \Rightarrow \mathbf{M} \approx \mathbf{I} \Rightarrow$
  for small τ_k, solution is very fast,
- for increasing τ_k, solution more expensive,
- choice of τ_k ?

Linear algebra problem

- iterative solver necessary for industrial applications
  $\Rightarrow$ restarted GMRES with block diagonal preconditioning
- L^2-orthogonal basis of $\mathbf{S}_h \Rightarrow \mathbf{M} \approx \mathbf{I} \Rightarrow$
  for small τ_k, solution is very fast,
- for increasing τ_k, solution more expensive,
- choice of τ_k ?

Linear algebra problem

- iterative solver necessary for industrial applications
  $\Rightarrow$ restarted GMRES with block diagonal preconditioning
- L^2-orthogonal basis of $\mathbf{S}_h \Rightarrow \mathbf{M} \approx \mathbf{I} \Rightarrow$
  for small τ_k, solution is very fast,
- for increasing τ_k, solution more expensive,
- choice of τ_k ?

Linear algebra problem

- iterative solver necessary for industrial applications
  $\Rightarrow$ restarted GMRES with block diagonal preconditioning
- L^2-orthogonal basis of $\mathbf{S}_h \Rightarrow \mathbf{M} \approx \mathbf{I} \Rightarrow$
  for small τ_k, solution is very fast,
- for increasing τ_k, solution more expensive,
- choice of τ_k ?

Linear algebra problem

- iterative solver necessary for industrial applications
  $\Rightarrow$ restarted GMRES with block diagonal preconditioning
- L^2-orthogonal basis of $\mathbf{S}_h \Rightarrow \mathbf{M} \approx \mathbf{I} \Rightarrow$
  for small τ_k, solution is very fast,
- for increasing τ_k, solution more expensive,
- choice of τ_k ?

Choice of the time step

- ABDF – adaptive BDF [Dolejší, Kůs, IJNME]
- two n-step BDF of the same order of accuracy,

$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{I}} \mathbf{w}_{k-l}^{\mathrm{I}} = \tau_k F(\mathbf{w}_k^{\mathrm{I}}),$$
$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{II}} \mathbf{w}_{k-l}^{\mathrm{II}} = \frac{\tau_k}{2} \left(F(\mathbf{w}_k^{\mathrm{II}}) + F(\mathbf{w}_{k-1}^{\mathrm{II}}) \right), \qquad (4)$$

- from $\|\mathbf{w}_{k-l}^{\mathrm{I}} - \mathbf{w}_{k-l}^{\mathrm{II}}\|$ we estimate local discretization error e_k
- propose a time step such that $e_k \approx \omega, \quad \omega > 0$

Choice of the time step

- ABDF – adaptive BDF [Dolejší, Kůs, IJNME]
- two n-step BDF of the same order of accuracy,

$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{I}} \mathbf{w}_{k-l}^{\mathrm{I}} = \tau_k F(\mathbf{w}_k^{\mathrm{I}}),$$

$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{II}} \mathbf{w}_{k-l}^{\mathrm{II}} = \frac{\tau_k}{2} \left(F(\mathbf{w}_k^{\mathrm{II}}) + F(\mathbf{w}_{k-1}^{\mathrm{II}}) \right), \qquad (4)$$

- from $\|\mathbf{w}_{k-l}^{\mathrm{I}} - \mathbf{w}_{k-l}^{\mathrm{II}}\|$ we estimate local discretization error e_k
- propose a time step such that $e_k \approx \omega, \quad \omega > 0$

Choice of the time step

- ABDF – adaptive BDF [Dolejší, Kůs, IJNME]
- two n-step BDF of the same order of accuracy,

$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{I}} \mathbf{w}_{k-l}^{\mathrm{I}} = \tau_k F(\mathbf{w}_k^{\mathrm{I}}),$$
$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{II}} \mathbf{w}_{k-l}^{\mathrm{II}} = \frac{\tau_k}{2}\left(F(\mathbf{w}_k^{\mathrm{II}}) + F(\mathbf{w}_{k-1}^{\mathrm{II}})\right), \qquad (4)$$

- from $\|\mathbf{w}_{k-l}^{\mathrm{I}} - \mathbf{w}_{k-l}^{\mathrm{II}}\|$ we estimate local discretization error e_k
- propose a time step such that $e_k \approx \omega, \quad \omega > 0$

Choice of the time step

- ABDF – adaptive BDF [Dolejší, Kůs, IJNME]
- two n-step BDF of the same order of accuracy,

$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{I}} \mathbf{w}_{k-l}^{\mathrm{I}} = \tau_k F(\mathbf{w}_k^{\mathrm{I}}),$$
$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{II}} \mathbf{w}_{k-l}^{\mathrm{II}} = \frac{\tau_k}{2} \left(F(\mathbf{w}_k^{\mathrm{II}}) + F(\mathbf{w}_{k-1}^{\mathrm{II}}) \right), \qquad (4)$$

- from $\|\mathbf{w}_{k-l}^{\mathrm{I}} - \mathbf{w}_{k-l}^{\mathrm{II}}\|$ we estimate local discretization error e_k
- propose a time step such that $e_k \approx \omega, \quad \omega > 0$

Choice of the time step

- ABDF – adaptive BDF [Dolejší, Kůs, IJNME]
- two n-step BDF of the same order of accuracy,

$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{I}} \mathbf{w}_{k-l}^{\mathrm{I}} = \tau_k F(\mathbf{w}_k^{\mathrm{I}}),$$

$$\sum_{l=0}^{n} \alpha_{n,l}^{\mathrm{II}} \mathbf{w}_{k-l}^{\mathrm{II}} = \frac{\tau_k}{2} \left(F(\mathbf{w}_k^{\mathrm{II}}) + F(\mathbf{w}_{k-1}^{\mathrm{II}}) \right), \qquad (4)$$

- from $\|\mathbf{w}_{k-l}^{\mathrm{I}} - \mathbf{w}_{k-l}^{\mathrm{II}}\|$ we estimate local discretization error e_k
- propose a time step such that $e_k \approx \omega$, $\quad \omega > 0$

Parallel implementation

1 Introduction

2 Parallel implementation

3 Outlook

Adgfem

- ADGFEM - software implementing ABDF - DGFEM
- Aim: parallel implementation
- Current release not prepared for parallel code
- We are doing initial research

Adgfem

- ADGFEM - software implementing ABDF - DGFEM
- Aim: parallel implementation
- Current release not prepared for parallel code
- We are doing initial research

Adgfem

- ADGFEM - software implementing ABDF - DGFEM
- Aim: parallel implementation
- Current release not prepared for parallel code
- We are doing initial research

Adgfem

- ADGFEM - software implementing ABDF - DGFEM
- Aim: parallel implementation
- Current release not prepared for parallel code
- We are doing initial research

Adgfem

- ADGFEM - software implementing ABDF - DGFEM
- Aim: parallel implementation
- Current release not prepared for parallel code
- We are doing initial research

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results

- e.g. solving matrix with size n = 214656

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results
- e.g. solving matrix with size n = 214656

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results

- e.g. solving matrix with size n = 214656

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results

- e.g. solving matrix with size $n = 214656$

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results

- e.g. solving matrix with size n = 214656

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results

- e.g. solving matrix with size n = 214656

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results

- e.g. solving matrix with size n = 214656

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results

- e.g. solving matrix with size n = 214656

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Solvability testing

- used PETSc (Portable, Extensible Toolkit for Scientific Computation) library
- used MUMPS solver and got surprising results

- e.g. solving matrix with size n = 214656

# processors	1	2	4	8
computational time	120 s	19645 s	4999 s	1501 s

Graph

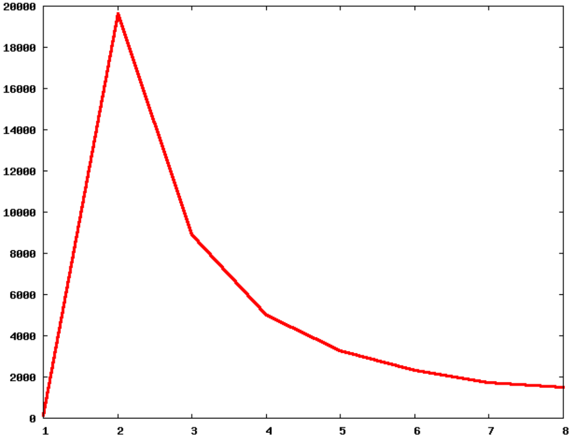

Figure: Dependency of computational time on number of processors

Parallelization

- linear algebra solver ≈ 95% of CPU
  ⇒ we focus on its parallelization
- PETSc library gives easy-to-use interface for solving matrices
  ⇒ only problem are matrix data structures
- Our most difficult task in this release: transformation of the "single-processor" matrices into PETSc structures.

Parallelization

- linear algebra solver $\approx$ 95% of CPU
  $\Rightarrow$ we focus on its parallelization
- PETSc library gives easy-to-use interface for solving matrices
  $\Rightarrow$ only problem are matrix data structures
- Our most difficult task in this release: transformation of the "single-processor" matrices into PETSc structures.

Parallelization

- linear algebra solver $\approx$ 95% of CPU
  $\Rightarrow$ we focus on its parallelization
- PETSc library gives easy-to-use interface for solving matrices
  $\Rightarrow$ only problem are matrix data structures
- Our most difficult task in this release: transformation of the "single-processor" matrices into PETSc structures.

Parallelization

- linear algebra solver $\approx$ 95% of CPU
  $\Rightarrow$ we focus on its parallelization
- PETSc library gives easy-to-use interface for solving matrices
  $\Rightarrow$ only problem are matrix data structures
- Our most difficult task in this release: transformation of the "single-processor" matrices into PETSc structures.

Parallelization

- linear algebra solver $\approx$ 95% of CPU
  $\Rightarrow$ we focus on its parallelization
- PETSc library gives easy-to-use interface for solving matrices
  $\Rightarrow$ only problem are matrix data structures
- Our most difficult task in this release: transformation of the "single-processor" matrices into PETSc structures.

Parallelization

- linear algebra solver $\approx$ 95% of CPU
  $\Rightarrow$ we focus on its parallelization
- PETSc library gives easy-to-use interface for solving matrices
  $\Rightarrow$ only problem are matrix data structures
- Our most difficult task in this release: transformation of the "single-processor" matrices into PETSc structures.

Parallelization

- linear algebra solver $\approx$ 95% of CPU
  $\Rightarrow$ we focus on its parallelization
- PETSc library gives easy-to-use interface for solving matrices
  $\Rightarrow$ only problem are matrix data structures
- Our most difficult task in this release: transformation of the "single-processor" matrices into PETSc structures.

Current state

- The linear solver in Adgfem is parallel
- Some of the structures are parallel
- Matrix data are not paralell yet

Current state

- The linear solver in Adgfem is parallel
- Some of the structures are parallel
- Matrix data are not paralell yet

Current state

- The linear solver in Adgfem is parallel
- Some of the structures are parallel
- Matrix data are not paralell yet

Current state

- The linear solver in Adgfem is parallel
- Some of the structures are parallel
- Matrix data are not paralell yet

MUMPS - computational time

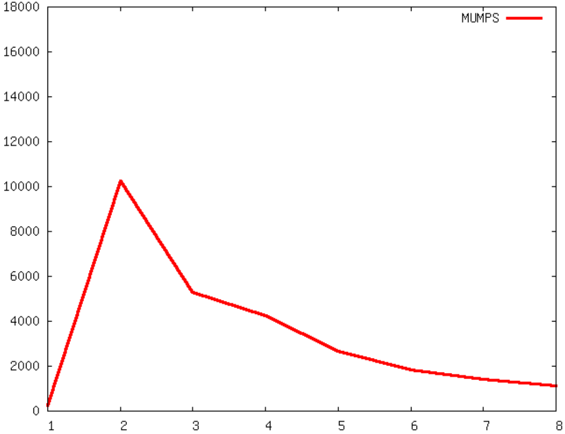

Figure: Dependency of computational time on number of processors

Computational time - graph

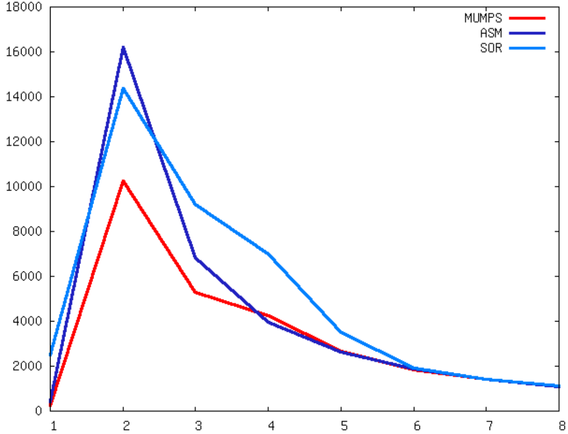

Figure: Dependency of computational time on number of processors

Computational time - survey

# processors	1	2	4	8
GMRES - SOR	2504 s	14360 s	7011 s	1105 s
GMRES - ASM	434 s	16214 s	3939 s	1103 s
MUMPS	252 s	4259 s	4259 s	1106 s

Requested memory - graph

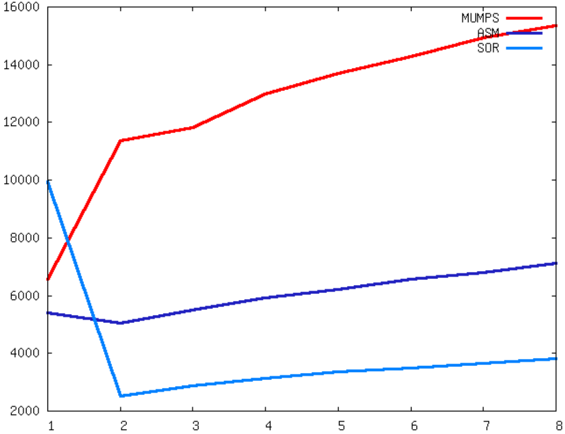

Figure: Dependency of requested memory on number of processors

Used memory - graph

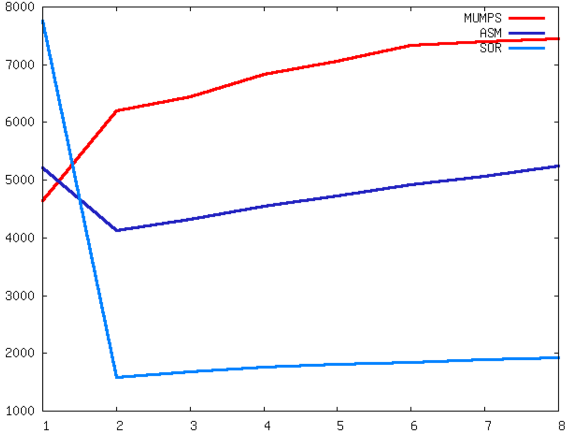

Figure: Dependency of used memory on number of processors

Outlook

1 Introduction

2 Parallel implementation

3 Outlook

Outlook

Outlook

- full parallelization
- *hp*-adaptation
- extension to 3D.

Outlook

Outlook

- full parallelization
- *hp*-adaptation
- extension to 3D.

Outlook

Outlook

- full parallelization
- *hp*-adaptation
- extension to 3D.

Outlook

Outlook

- full parallelization
- *hp*-adaptation
- extension to 3D.

Thank you for your attention!