# Co-design of time stepping algorithms for large aerodynamic simulations

AFOSR BRI 12-2640-06

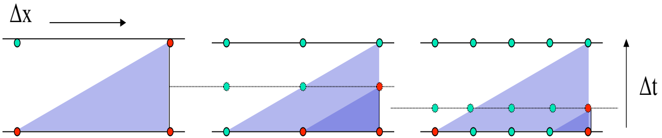
Adrian Sandu[1], Paul Tranquilli[1], Hong Zhang[1] and Ross Glandon[1]

[1] Computational Science Laboratory (CSL)
Department of Computer Science
Virginia Tech

February 7, 2014
AFOSR Workshop

# Challenges to solving large evolutionary PDEs and co-design solution approaches I

1. Explicit time stepping: simple, scalable, CFL bounded




2. Implicit time integration:
   - Unconditionally stable $\rightarrow$ step size determined by accuracy only
   - Huge nonlinear systems coupling all variables in the model at each time step
   - Error estimation and step size control lead to additional data dependencies
3. Our algorithmic co-design goals:
   - Identify and use minimal amount of implicitness
   - Use only operations that are scalable/amenable to acceleration

# Challenges to solving large evolutionary PDEs and co-design solution approaches II

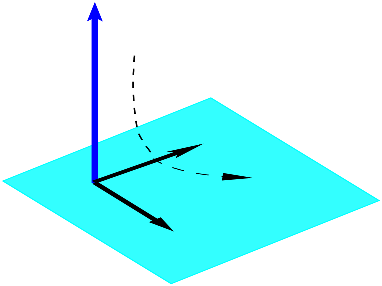


**Figure :** Solution approach 1: separate the small stiff subspace from the non-stiff subspace and use implicitness in the stiff subspace only: ROK, EXPK methods (Paul Tranquilli)

# Challenges to solving large evolutionary PDEs and co-design solution approaches III

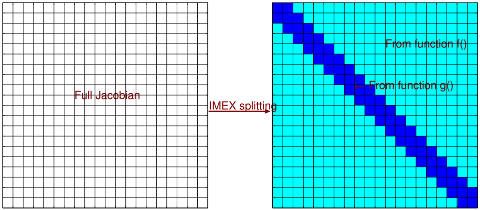



**Figure :** Solution approach 2: $y' = f(t, y) + g(t, y)$: Separate the stiff processes from the non-stiff processes and use implicitness to treat the stiff processes only: IMEX methods (Hong Zhang)

# Challenges to solving large evolutionary PDEs and co-design solution approaches IV

Solution approach 3: use highly scalable Jacobian-vector operations for efficient accelerated and distributed implementation (Ross Glandon)

- Parallel Burgers ODE function (local computation):

$$f_n^{i_p:j_p} = \frac{1}{2\Delta x} \begin{bmatrix} \left(\mathbf{y}_n^{i_p-1}\right)^2 - \left(\mathbf{y}_n^{i_p+1}\right)^2 \\ \left(\mathbf{y}_n^{i_p:j_p-2}\right)^2 - \left(\mathbf{y}_n^{i_p+2:j_p}\right)^2 \\ \left(\mathbf{y}_n^{j_p-1}\right)^2 - \left(\mathbf{y}_n^{j_p+1}\right)^2 \end{bmatrix}$$

- Scalable Jacobian-vector product (local computation):

$$(J_n\, v)^{i_p:j_p} = \frac{1}{\Delta x} \begin{bmatrix} \mathbf{y}_n^{i_p-1}\, v^{i_p-1} - \mathbf{y}_n^{i_p+1}\, v^{i_p+1} \\ \mathbf{y}_n^{i_p:j_p-2}\, v^{i_p:j_p-2} - \mathbf{y}_n^{i_p+2:j_p}\, v^{i_p+2:j_p} \\ \mathbf{y}_n^{j_p-1}\, v^{j_p-1} - \mathbf{y}_n^{j_p+1}\, v^{j_p+1} \end{bmatrix}$$

# Rosenbrock methods require the solution of linear systems only

- Initial value problem (semi-discrete PDE)

$$y'(t) = f(y), \quad y(t_0) = y_0, \quad t_0 \le t \le t_{\mathrm{F}}\,, \quad y(t), f(y) \in \mathbb{R}^N.$$

- Solution by an $s$-stage Rosenbrock method:

$$(\mathbf{I} - h\gamma \mathbf{J}_n)\, k_i = h\, f\left( y_n + \sum_{j=1}^{i-1} \alpha_{ij} k_j \right) + h\mathbf{J}_n \sum_{j=1}^{i-1} \gamma_{ij} k_j\,,$$

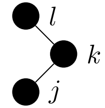
$$y_1 = y_0 + \sum_{j=1}^{s} b_i k_i\,.$$

- The Jacobian matrix, $\mathbf{J}_n = \partial f / \partial y\,|_{y=y_n}$ appears explicitly.

# Rosenbrock-W order conditions

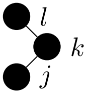
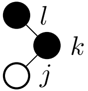
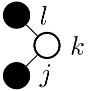
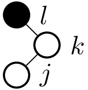
- $TW$-trees (bi-colored, leaves full, empty vertices singly branched)
- Full nodes $\sim$ exact derivatives, empty nodes $\sim$ $\mathbf{A}$.

$T$-tree

| | |
|---|---|
|   | $f^J_K f^K_L f^L$ |

$\longrightarrow$

$TW$-trees

| | |
|---|---|
|   | $f^J_K f^K_L f^L$ |
|   | $\mathbf{A}_{JK} f^K_L f^L$ |
|   | $f^J_K \mathbf{A}_{KL} f^L$ |
|   | $\mathbf{A}_{JK} \mathbf{A}_{KL} f^L$ |

# Definition: ROK method in autonomous form

Arnoldi: compute $\mathbf{H}$ and $\mathbf{V}$ for $\mathcal{K}_M\left(\mathbf{J}_n, f_n\right)$

**for** i = 1 **to** s

$$F_i = f\left(y_n + \sum_{j=1}^{i-1} \alpha_{ij} k_j\right)$$

$$\psi_i = \mathbf{V}^T f_i$$

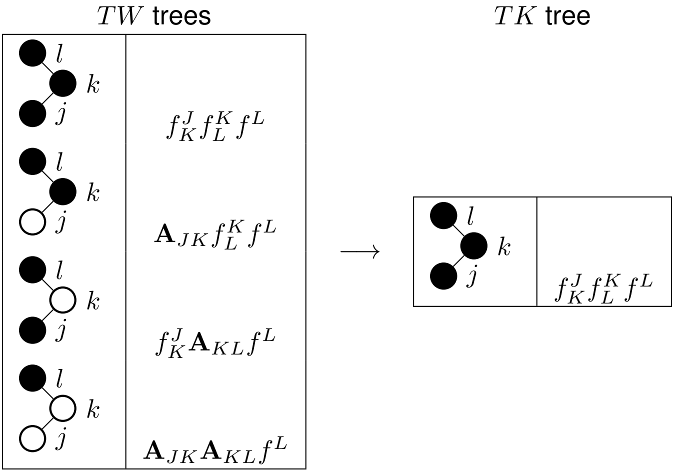
$$\lambda_i = \left(\mathbf{I}_{M\times M} - h\gamma\mathbf{H}\right)^{-1}\left(h\psi_i + h\mathbf{H}\sum_{j=1}^{i-1}\gamma_{ij}\lambda_j\right)$$

$$k_i = \mathbf{V}\lambda_i + h\left(F_i - \mathbf{V}\phi_i\right)$$

**end for** i

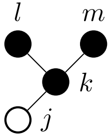
$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i\, k_i$$

# The Krylov approximation property reduces the set of relevant trees considerably



# ROK methods

- ROK conditions up to order three $\equiv$ ROS conditions
- There is one additional $TK$-tree and ROK condition for order four

| | | |
|---|---|---|
|   | $A_{JK} f^{K}_{LM} f^{L} f^{M}$ | $\sum b_j \gamma_{jk} \alpha_{km} \alpha_{kl} \quad = \quad 0$ |

## Theorem (Type 1 order conditions)

*A Rosenbrock-K method of type 1 has order $p$ iff the underlying Krylov space has dimension $M \geq p$, and the following order conditions hold:*

$$\sum_j b_j\, \phi_j(t) = \frac{1}{\gamma(t)} \quad \forall\, t \in T \;\; \textit{with } \rho(t) \leq p\,,$$

$$\sum_j b_j\, \phi_j(t) = 0 \quad \forall\, t \in TK \backslash T \;\; \textit{with } \rho(t) \leq p\,.$$

# Convergence and Stability

For accuracy:

- $M$ is small and independent of problem size.

For stability:

- Intuitively $M$ should be sufficiently large such that the Krylov space contains the stiff subspace of the underlying problem (see also Weiner et al)
- How to automatically choose $M$ so that the method is stable is a topic of ongoing work.

# Definition: LIKE method in autonomous form

Arnoldi: compute $\mathbf{H}$ and $\mathbf{V}$ for $\mathcal{K}_M\left(\mathbf{J}_n, f_n\right)$

**for** i = 1 **to** s

$$F_i = f\left(y_n + \sum_{j=1}^{i-1} \alpha_{ij} k_j\right)$$

$$\psi_i = \mathbf{V}^T f_i$$

$$\lambda_i = \varphi(h\gamma\mathbf{H})\left(h\psi_i + h\mathbf{H}\sum_{j=1}^{i-1} \gamma_{ij}\lambda_j\right)$$

$$k_i = \mathbf{V}\lambda_i + h\left(F_i - \mathbf{V}\phi_i\right)$$

**end for** i

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i\, k_i$$

# ROK methods outperform traditional ROS solvers on a two dimensional shallow water test problem

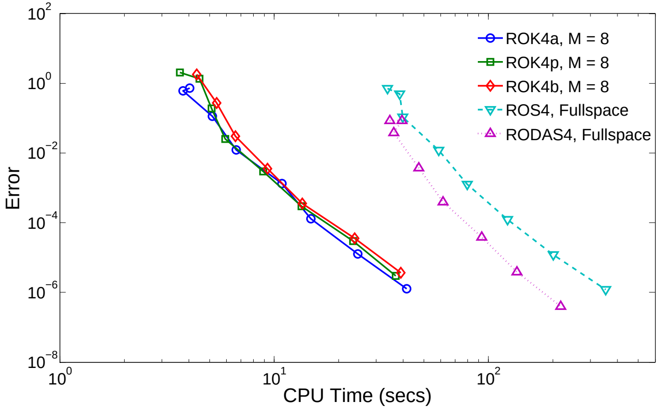



**Figure :** Performance comparison on shallow water equations using centered finite differences on a $32 \times 32$ cartesian grid, $N = 3072$.

# LIKE methods outperform traditional exponential solvers on a two dimensional shallow water test problem

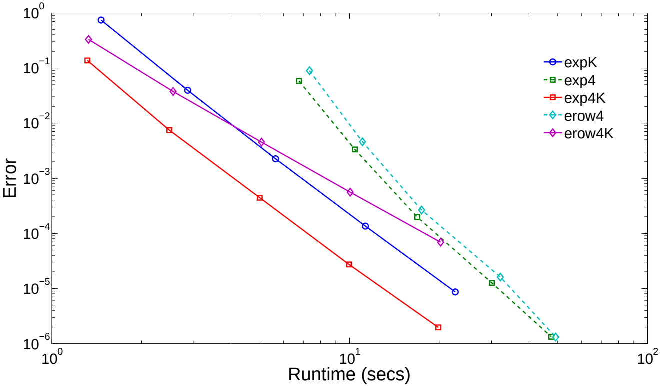
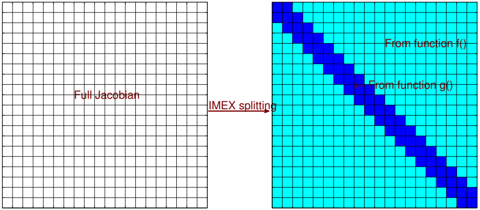



**Figure :** Performance comparison on shallow water equations using centered finite differences on a $32 \times 32$ cartesian grid, $N = 3072$.

# IMplicit-EXplicit time stepping schemes I

- Challenges:
  - **Stiff problems** Stiffness results from widely varying time scales, i.e., some components of the solution decay much more rapidly than others
  - **Explicit methods** are efficient for nonstiff problems; require extremely small time steps for stiff problems
  - **Implicit methods** allow for large time steps for stiff problems; computationally expensive
- One way to attack stiff problems efficiently: **IMEX method** partition the system into two part based on stiffness $y' = f(t, y) + g(t, y)$; treat stiff part implicitly while nonstiff part explicitly

# IMplicit-EXplicit time stepping schemes II




- Existing IMEX families:
  - IMEX Linear Multistep Method (poor stability)
  - IMEX Runge-Kutta methods (order reduction)
- Goal: to develop new IMEX Methods with several properties:
  - no order reduction
  - good stability
  - ...

# IMEX DIMSIM

A two-way partitioned DIMSIM: $(\widehat{\mathbf{A}}, \widehat{\mathbf{B}})$ implicit, $(\mathbf{A}, \mathbf{B})$ explicit

$$
\begin{aligned}
Y_i &= h\left(\sum_{j=1}^{i-1} a_{i,j}\, f(Y_j) + \sum_{j=1}^{i} \widehat{a}_{i,j}\, g(Y_j)\right) + y_i^{[n-1]}, \quad i = 1, \ldots, s\,, \\
y_i^{[n]} &= h\left(\sum_{j=1}^{s} b_{i,j}\, f(Y_j) + \sum_{j=1}^{s} \widehat{b}_{i,j}\, g(Y_j)\right) + \sum_{j=1}^{r} v_{i,j}\, y_j^{[n-1]}, \quad i = 1, \ldots, r\,.
\end{aligned}
$$

Derivation: Assume

$$
y = x + z\,, x' = \tilde{f}(x, z) = f(x + z)\,, z' = \tilde{g}(x, z) = g(x + z)\,,
$$

we do not need to know what $x$ and $z$ are. It works as if the combined state $y$ is advanced through integration.
Starting procedure: Approximate $h^k x^{(k)}(t_0)$, $h^k z^{(k)}(t_0)$, using finite differences on small step solutions.

# Properties of IMEX DIMSIM

- High stage order Order. Order $p$, stage order $q$, number of external stages $r$, number of internal stages $s$ are related by $p = q = r = s$.
- Implicit part is L-stable and constrained explicit stability region is maximized using optimization technique. DIMSIMs are constructed with Runge-Kutta stability.
- No additional coupling condition.

## Theorem (Zhang and Sandu, 2012)

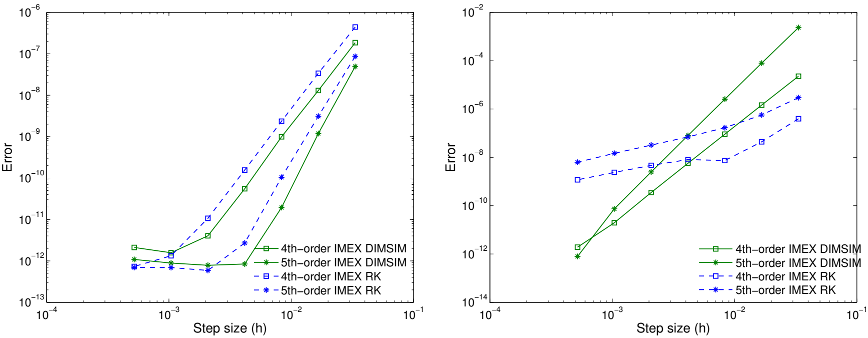
- *Partitioned DIMSIM has order* $p$ *and stage order* $q = p$ $\Updownarrow$ *each individual method has order* $p$ *and stage order* $q = p$*.*
- *Partitioned DIMSIM has order* $p$ *and stage order* $q = p - 1$ $\Updownarrow$ *each constituent method has order* $p$ *and stage order* $q = p - 1$*.*

# Avoid order reduction

Consider the van der Pol equation (Boscarino, 2007)

$$\frac{d}{dt}\begin{bmatrix} y \\ z \end{bmatrix} = \underbrace{\begin{bmatrix} z \\ 0 \end{bmatrix}}_{f(y,z)} + \underbrace{\begin{bmatrix} 0 \\ ((1-y^2)z - y)/\epsilon \end{bmatrix}}_{g(y,z)}, \quad 0 \le t \le 0.55139$$

$$y(0) = 2, \quad z(0) = -\frac{2}{3} + \frac{10}{81}\epsilon - \frac{292}{2187}\epsilon^2 - \frac{1814}{19683}\epsilon^3 + \mathcal{O}(\epsilon^4)\,.$$




(a) nonstiff case $\epsilon = 10^{-1}$ (b) stiff case $\epsilon = 10^{-5}$

# Gravity waves I

GMSH-DG code (UCLouvain): discontinuous Galerkin method in space discretization

Governed by the compressible Euler equation

$$\begin{aligned}\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) &= 0\\ \frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u} + p\mathbf{I}) &= -\rho g \widehat{\mathbf{e}}_{\mathbf{z}}\\ \frac{\partial \rho\theta}{\partial t} + \nabla \cdot (\rho\theta \mathbf{u}) &= 0\end{aligned}$$

$\rho$ : density
$\mathbf{u}$ : velocity
$\theta$ : potential temperature
$\mathbf{I}$ : a $2 \times 2$ identity matrix
$p$ : pressure (linearly related to $\rho\theta$)
The prognostic variables are $\rho, \rho\mathbf{u}, \rho\theta$

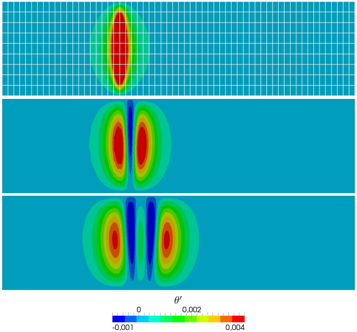
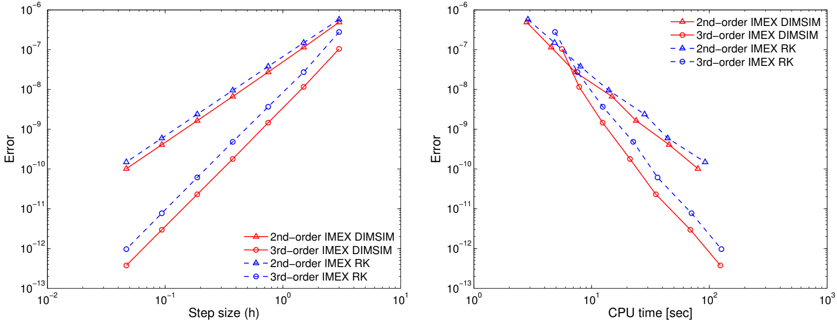



**Figure :** Evolution of the gravity wave: perturbation of the potential temperature at the initial time (top), after 450 seconds (middle) and after 900 seconds (bottom).

# Gravity waves III




(a) Convergence

(b) Work-precision diagram

# Parallelizing ROK methods I

We target the Rosenbrock-Krylov (ROK) class of methods.

- Implicit method
  - Based on Rosenbrock implicit methods
  - Uses a Krylov subspace method
- Inexpensive
  - Requires only a linear solve
  - Operates in a reduced space
  - Matrix-free

# Parallelizing ROK methods II

Sources of ROK methods' advantages:

- Linearization inherited from Rosenbrock methods.
- Accuracy is not required in the solution to the linear system.
- Uses a Krylov subspace approximation to the Jacobian of the ODE.
- Approximates Jacobian vector products using a finite difference.

# Notes about the multicore results

Experiments were performed on the gravity waves problem.

Three types of integrators were tested:

- ERK: an explicit Runge-Kutta method
- DIRK: a diagonally implicit Runge-Kutta method
- ROK: a Rosenbrock-Krylov method

Speedups are calculated using a serial implementation as a baseline.

Tests were performed on a quad socket machine using AMD Magny-Cours CPUs with a total of 48 cores.

# Runtime for multicore parallel solvers on the gravity waves problem

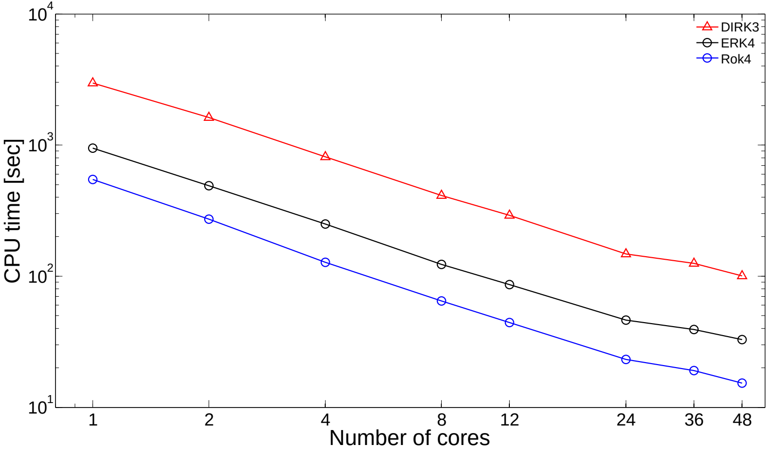



**Figure :** Solver runtimes for various core counts.

# Slowdown for multicore parallel solvers on the gravity waves problem

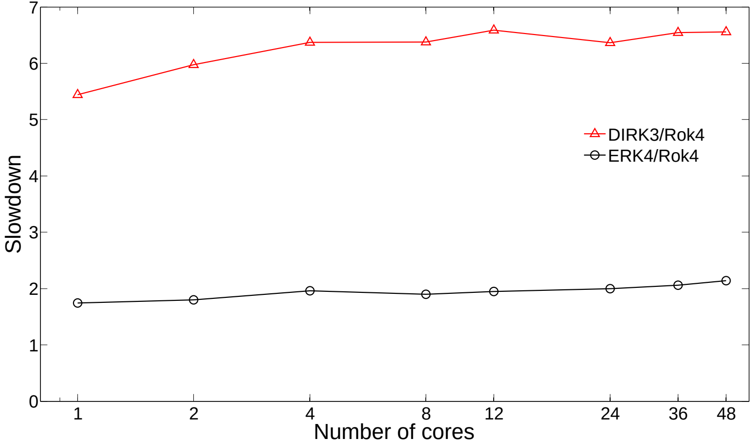



**Figure :** Slowdown of DIRK and ERK methods compared to the ROK solver.

# Parallel efficiency for multicore parallel solvers on the gravity waves problem

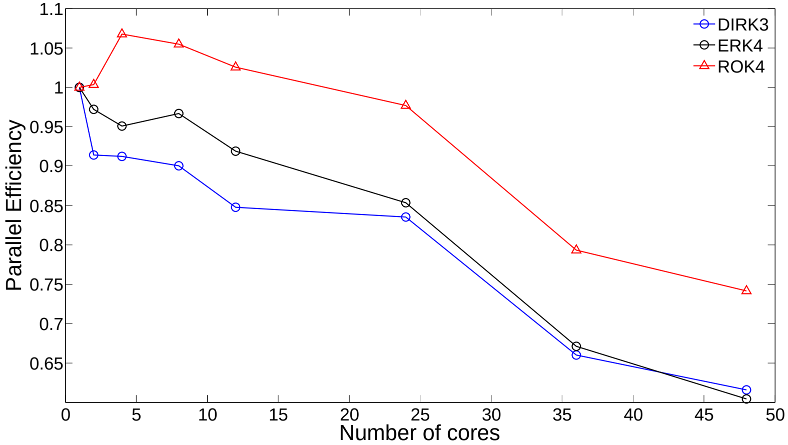



**Figure :** Parallel efficiency of the different solvers.

# Notes about the GPU results

Experiments were performed on the shallow water equations.
Two Arnoldi implementations were tested:

- cuKrylov: Basic cuBLAS implementation
- gtKrylov: Our optimized implementation

Speedups are calculated using a serial implementation as a baseline.
Tests were performed on a AMD Magny-Cours CPU and an NVIDIA Quadro 4000 GPU.

# Right hand side speedup for the shallow water equations problem on GPUs

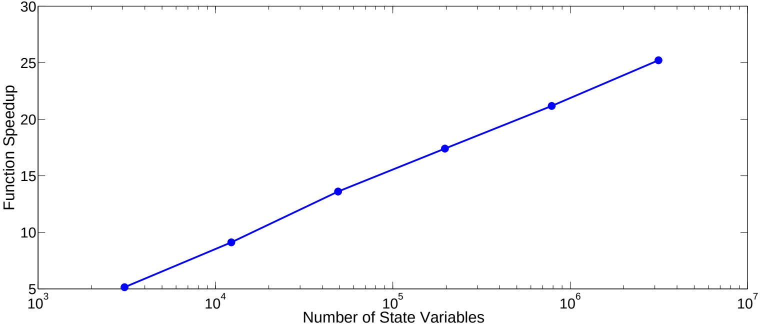



**Figure :** GPU RHS speedup over serial CPU.

# Total solver speedup for the shallow water equations problem on GPUs

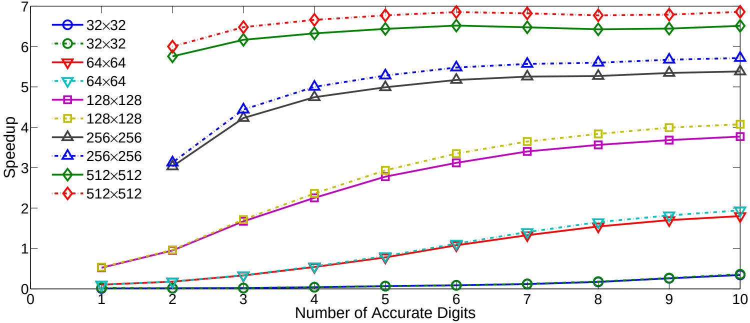
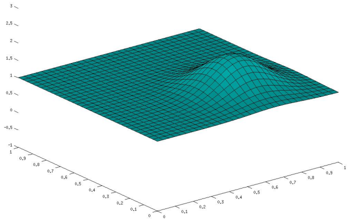
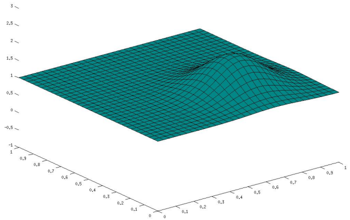



**Figure :** GPU solver speedup over serial CPU.

# Speedup animation for the shallow water equations problem on GPUs



(a) GPU solution speed.



(b) CPU solution speed.