Imperial College London

# Recent functionality and efficiency enhancements in Nektar++

Gabriele Rocco

Imperial College London

# Outline

1) Introduction.

2) Navier-Stokes equations in cylindrical coordinates.

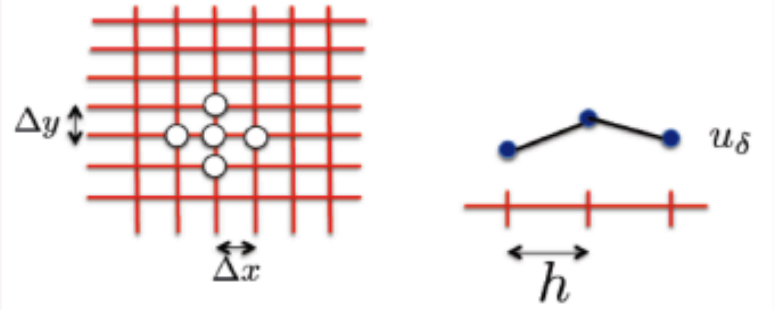
3) Parallelisation of large scale eigenvalue solvers.

4) Generalisation of the boundary conditions.

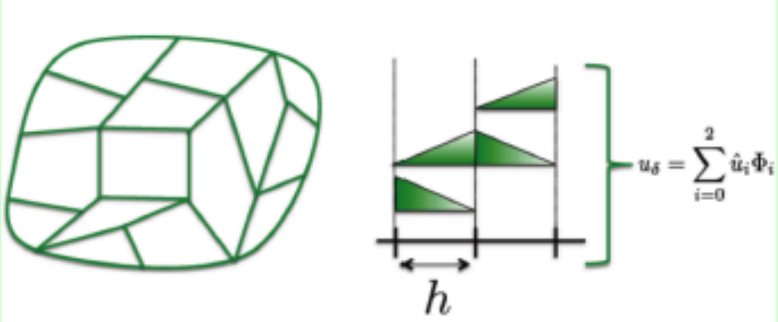
5) Conclusions and future developments.

# Introduction: spectral/*hp* element method

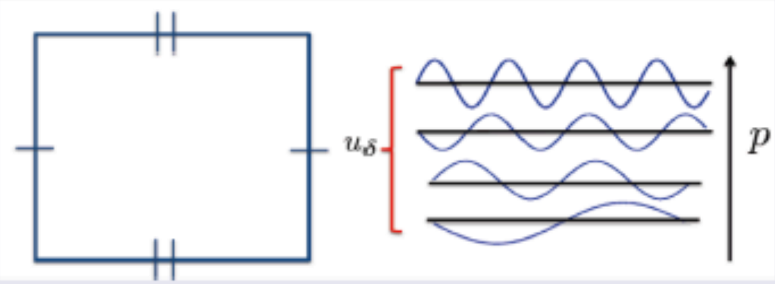
## Finite differences



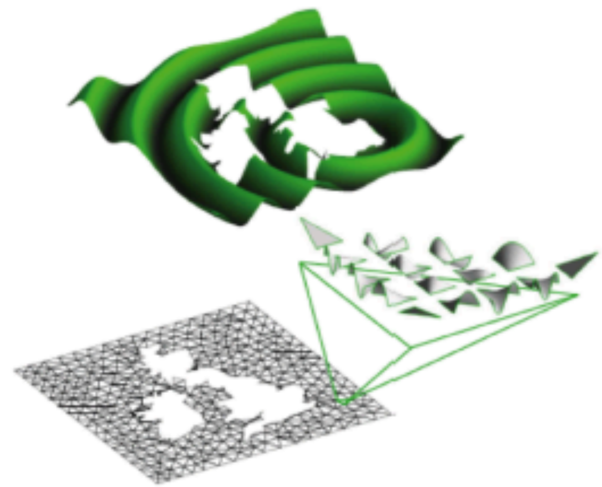

## Finite element methods



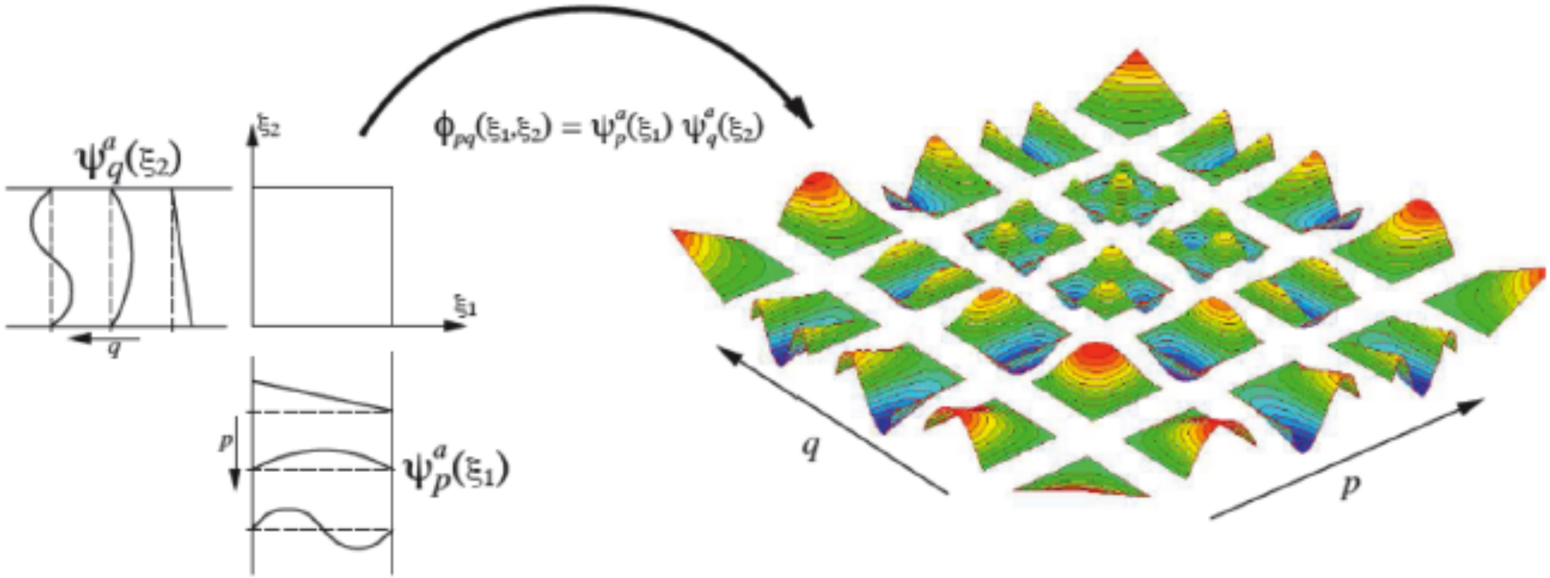

## Spectral method




## Spectral/hp element methods



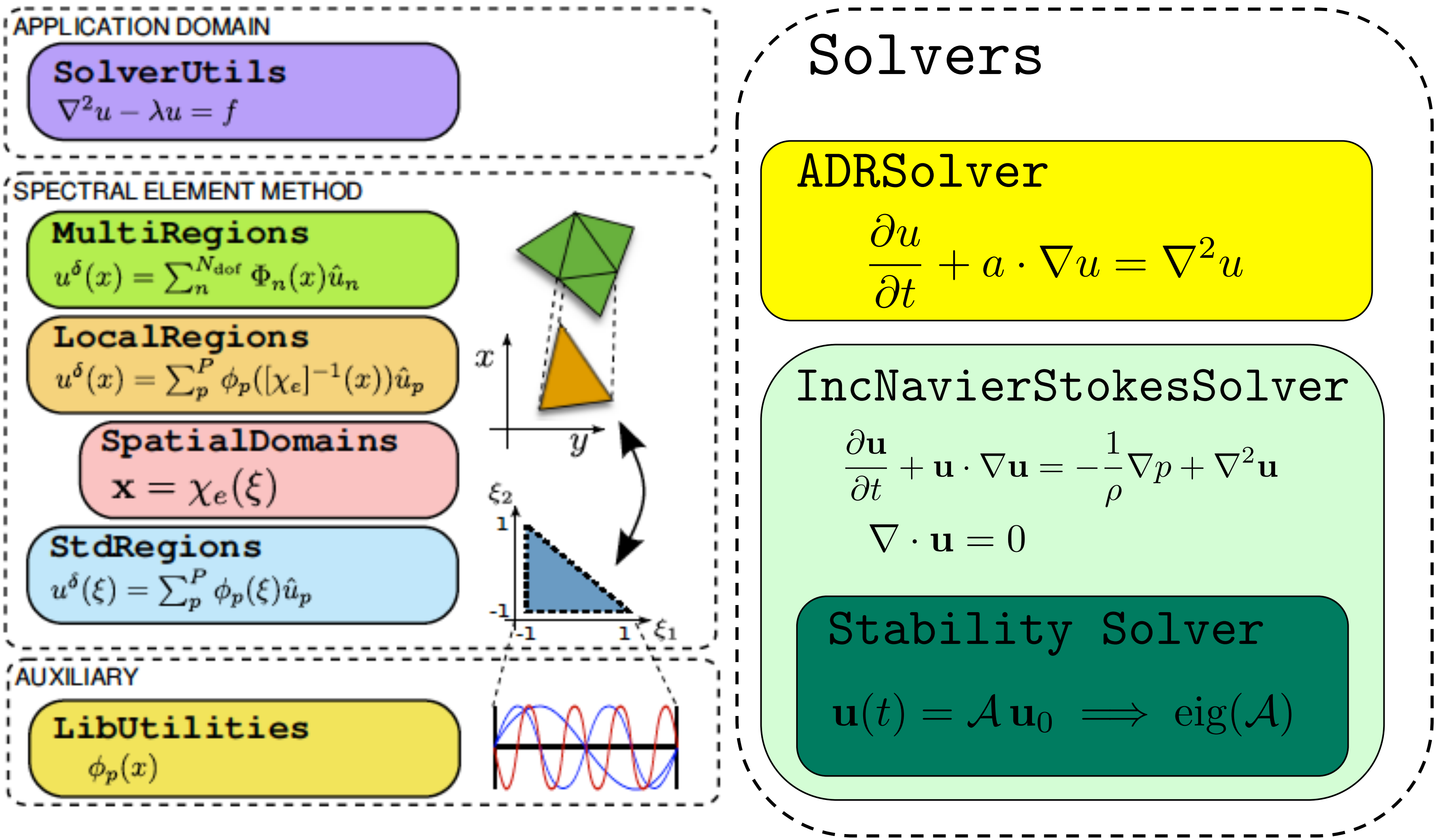
# Introduction: spectral/*hp* element method




- Exponential convergence.
- High geometric flexibility (suitable for complex geometries).
- Two types of refinements : geometric (*h*-type) and spectral (*p*-type)

Imperial College
London

# Introduction: Nektar++ framework



# Navier-Stokes equations in cylindrical coordinates

Cylindrical coordinates framework: $\mathbf{u} = \mathbf{u}(r, \theta, z, t)$

**GRADIENT:**

$$\nabla(\cdot) = \left(\frac{\partial \cdot}{\partial r}, \frac{1}{r}\frac{\partial \cdot}{\partial \theta}, \frac{\partial \cdot}{\partial z}\right)$$

**LAPLACIAN:**

$$\nabla^2(\cdot) = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial \cdot}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 \cdot}{\partial \theta^2} + \frac{\partial^2 \cdot}{\partial z^2}$$

**DIVERGENCE:**

$$\nabla \cdot \mathbf{u} = \frac{1}{r}\frac{\partial}{\partial r}(r u_r) + \frac{1}{r}\frac{\partial u_\theta}{\partial \theta} + \frac{\partial u_z}{\partial z}$$

# Navier-Stokes equations in cylindrical coordinates

Let us start simple... $\mathbf{u} = \mathbf{u}(r, z)$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial z} + v\frac{\partial u}{\partial r} = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \nu\left[\frac{\partial^2 u}{\partial z^2} + \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right)\right]$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial z} + v\frac{\partial v}{\partial r} = -\frac{1}{\rho}\frac{\partial p}{\partial r} + \nu\left[\frac{\partial^2 v}{\partial z^2} + \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial v}{\partial r}\right)\right]$$

$$\frac{\partial u}{\partial z} + \frac{1}{r}\frac{\partial}{\partial r}(rv) = 0$$

Imperial College London

# Navier-Stokes equations in cylindrical coordinates

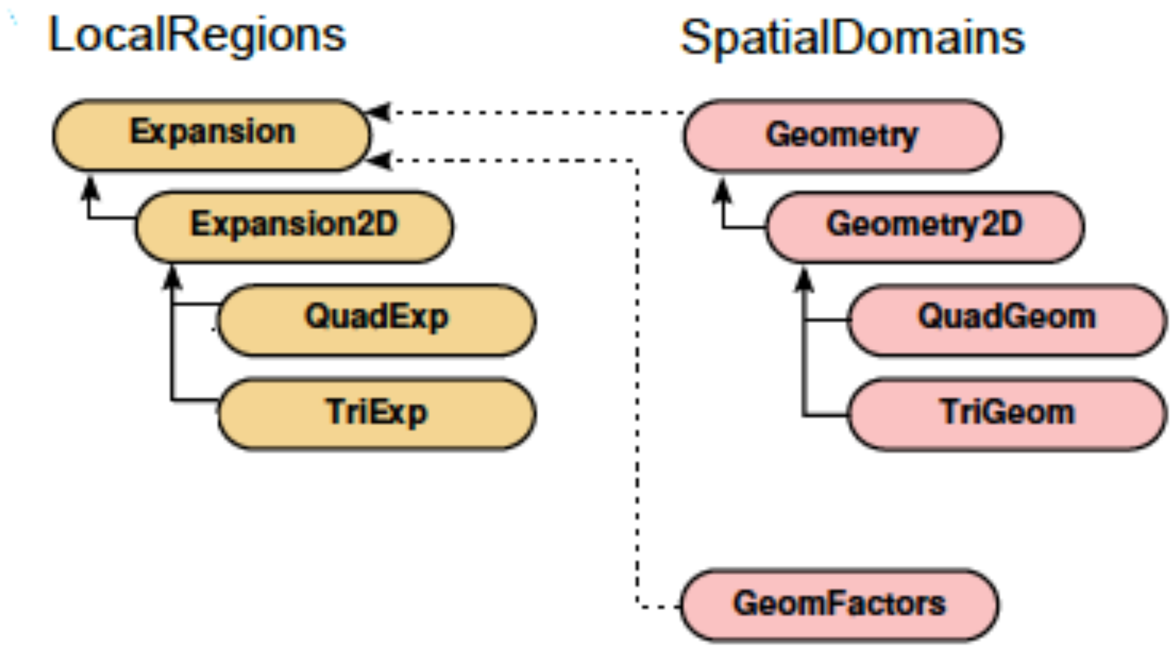
**Symmetrisation**: eliminate the singularity with respect to *r*

$$r\frac{\partial u}{\partial t} + r\left(u\frac{\partial u}{\partial z} + v\frac{\partial u}{\partial r}\right) = -\frac{r}{\rho}\frac{\partial p}{\partial z} + \nu r\frac{\partial^2 u}{\partial z^2} + \nu\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right)$$

$$r\frac{\partial v}{\partial t} + r\left(u\frac{\partial v}{\partial z} + v\frac{\partial v}{\partial r}\right) = -\frac{r}{\rho}\frac{\partial p}{\partial r} + \nu r\frac{\partial^2 v}{\partial z^2} + \nu\frac{\partial}{\partial r}\left(r\frac{\partial v}{\partial r}\right)$$

$$r\frac{\partial u}{\partial z} + \frac{\partial}{\partial r}(rv) = 0$$

# Navier-Stokes equations in cylindrical coordinates




**Cartesian:**
$$\int_{\Omega^e} u(x,y)dxdy = \int_{\Omega_{st}} u(\xi_1,\xi_2)|J|d\xi_1 d\xi_2$$

**Cylindrical:**
$$\int_{\Omega^e} ru(z,r)drdz = \int_{\Omega_{st}} u(\xi_1,\xi_2)\underbrace{r|J|}_{J_c}d\xi_1 d\xi_2$$

Imperial College London

# Navier-Stokes equations in cylindrical coordinates

## Simple implementation idea:

```
//GeomFactors.cpp

Vmath::Vsqrt(ptsTgt, &jac[0], 1, &jac[0], 1);
…

if(m_cylindrical)
{
      Vmath::Vmul(ptsTgt, &Radial[0],1,&jac[0],1,&jac[0],1);
}
```

**We treat the jacobian as a vector, similarly to the approach for a deformed element (evaluated at the quadrature point).**

# Navier-Stokes equations in cylindrical coordinates

## Session XML File set up:

1) Specification of the coordinate system: “**CARTESIAN**” (default), “**CYLINDRICAL**”

```
<NEKTAR>
<GEOMETRY DIM="2" SPACE="2" COORDINATE="CYLINDRICAL">
```

2) Specification of the axial boundary conditions:

```
//axial boundary conditions

<REGION REF="0">
   <A VAR="u" VALUE="0" />
   <A VAR="v" VALUE="0" />
   <A VAR="p" VALUE="0" />
  </REGION>
```

$$\frac{\partial u}{\partial r} = 0$$

$$v = 0$$

$$\frac{\partial p}{\partial r} = 0$$

# Navier-Stokes equations in cylindrical coordinates

## Test case: Kovasznay flow

- Laminar, steady flow behind a two-dimensional grid.
- Analytical solution in both Cartesian and cylindrical coordinates:

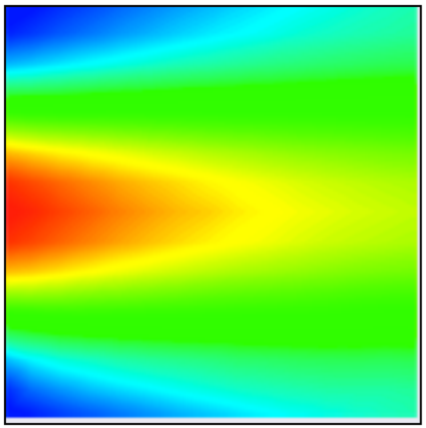
$$\begin{cases} u = 1 - \exp(\lambda z)\cos(2\pi r) \\ v = \dfrac{1}{2\pi}\lambda \exp(\lambda z)\sin(2\pi r) \\ p = \dfrac{1-\exp(\lambda z)}{2} \end{cases}$$

where

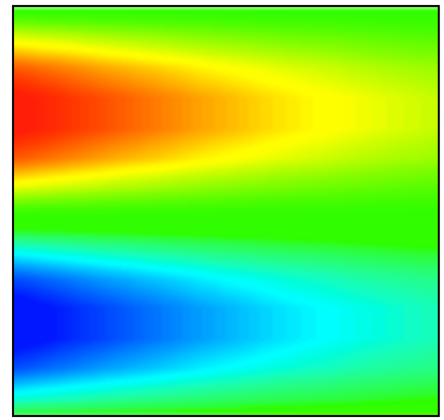
$$\begin{cases} \lambda = \dfrac{\mathrm{Re}}{2} - \sqrt{\dfrac{\mathrm{Re}^2}{4} + 4\pi^2} \\ \mathrm{Re} = 40 \end{cases}$$

# Navier-Stokes equations in cylindrical coordinates

$u$



$v$



$$\|\mathbf{u} - \mathbf{u}_{an}\|_2 \sim O(10^{-12})$$

Additional test case *in Nektar*++: laminar pipe flow.

Imperial College London

# Navier-Stokes equations in cylindrical coordinates

**Future extension: dependence on azimuthal direction**

$$\mathbf{u}(z,r,\theta,t) = \sum_{k=-\infty}^{\infty} \hat{\mathbf{u}}_k(z,r,t)\exp(ik\theta)$$

**The differential operators become:**

$$\nabla_k(\cdot) = \left(\frac{\partial\cdot}{\partial z}, \frac{\partial\cdot}{\partial r}, \frac{ik}{r}\cdot\right)$$

$$\nabla_k^2(\cdot) = \frac{\partial^2\cdot}{\partial z^2} + \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\cdot}{\partial r}\right) - \frac{k^2}{r^2}(\cdot)$$

**These expressions lead to a coupling of the *u* and *v* component s of the moment equations. A <u>diagonalisation</u> is required.**

# Parallelisation of large scale eigenvalue solvers

Given a dynamical system:

$$\dot{x}(t) = Ax(t)$$

**Definition (stability):**

A dynamical system is stable and $x_e$ is an equilibrium solution iff

$$\forall \varepsilon > 0 \, \exists \delta_\varepsilon \text{ such that } \forall t_0 : \|x(t_0) - x_e\| < \delta_\varepsilon \implies \|x(t) - x_e| < \varepsilon \; \forall t > t_0$$

**Theorem (linear stability):**

A dynamical system is linearly stable if:

$$\sup \Re[\sigma(A)] < 0$$

**This is the objective of Nektar++'s "stability solver"**

# Parallelisation of large scale eigenvalue solvers

## Arnoldi factorisation

If $\boldsymbol{A} \in \mathbb{C}^{n \times n}$:

$$\boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{V}_k\boldsymbol{H}_k + \beta_k \boldsymbol{v}_{k+1}\boldsymbol{e}_k^H$$

where $\boldsymbol{V}_k \in \mathbb{C}^{n \times k}$ has orthonormal columns and $\boldsymbol{H}_k \in \mathbb{C}^{k \times k}$ is upper Hessenberg matrix.

$$\lambda_k(A) \approx \lambda_{\max}(\mathbf{H}_k)$$

- Generate Krylov subspace:

$$\kappa = \{u_0, Au_0, A^2u_0, ..., A^{k-1}u_0\}$$

- QR Factorisation and computation of the Hessenberg matrix:

$$h_{i,j} = \frac{1}{r_{j,j}}\left(r_{i,j+1} - \sum_{m=0}^{j-1} h_{i,m}r_{m,j}\right)$$

- Use LAPACK to calculate the eigenvalue of the Hessenberg matrix.

Imperial College London

# Parallelisation of large scale eigenvalue solvers

## SolverUtils

### Advection

- Nonlinear
- Linearised
- Adjoint

### Driver

- Standard
- ModifiedArnoldi
- Arpack

## Solvers

### ADRSolver

$$\frac{\partial u}{\partial t} + a \cdot \nabla u = \nabla^2 u$$

### IncNavierStokesSolver

$$\frac{\partial \mathbf{u}}{\partial t} + N(\mathbf{u}) = -\frac{1}{\rho}\nabla p + \nabla^2 \mathbf{u}$$

$$\nabla \cdot \mathbf{u} = 0$$

### CompressibleFlowSolver

...

# Parallelisation of large scale eigenvalue solvers

```
//DriverModifiedArnoldi.cpp

...

//Krylov sequence.
for (i = 1; !converged && i <= m_kdim; ++i)
{
   //Krylov sequence.
   EV_update(Kseq[i-1], Kseq[i]);

  // Gram-Schmidt orthonormalisation
  alpha[i] = std::sqrt(Blas::Ddot(ntot, &Kseq[i][0], 1,
                                  &Kseq[i][0], 1));

m_comm->AllReduce(alpha[i], Nektar::LibUtilities::ReduceSum);
```

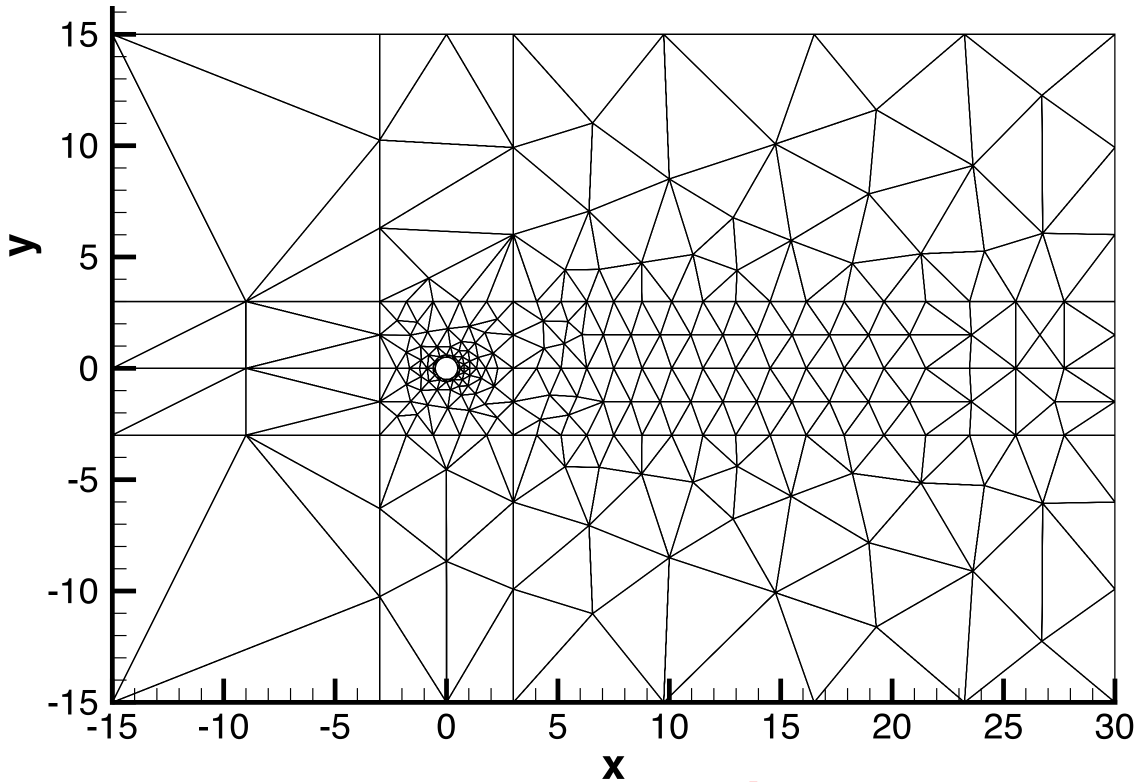
# Parallelisation of large scale eigenvalue solvers

```
…

Vmath::Smul(ntot, 1.0/alpha[i], Kseq[i], 1, Kseq[i], 1);

//Generate Hessenberg matrix and compute eigenvalues of it.

     EV_small(Tseq, ntot, alpha, i, zvec, wr, wi, resnorm);

//Test for convergence.

    converged = EV_test(i,i,zvec,wr,wi,resnorm,
                std::min(i,m_nvec),evlout,resid0);
    converged = max (converged, 0);
}
```

Imperial College London

# Parallelisation of large scale eigenvalue solvers

**Test case**: stability analysis of a flow past a 3D-cylinder




- $L_z$=5D;
- Re=40 (steady);
- k= 16;
- Fourier modes= 16;

**Time to convergence**

- **SERIAL** ~ 1day
- **PARALLEL (8processors)** ~ 10h

# Generalisation of the Boundary Conditions Framework

## Boundary Conditions supported in Nektar++:

1) **Dirichlet Boundary Conditions:**

$$\mathbf{u}(\mathbf{x},t)|_{\partial\Omega} = f(\mathbf{x},t)$$

2) **Neumann Boundary Conditions:**

$$\left.\frac{\partial \mathbf{u}}{\partial \mathbf{n}}(\mathbf{x},t)\right|_{\partial\Omega} = f(\mathbf{x},t)$$

3) **Robin Boundary Conditions:**

$$\left.\left[a\mathbf{u} + b\,\frac{\partial \mathbf{u}}{\partial \mathbf{n}}(\mathbf{x},t)\right]\right|_{\partial\Omega} = f(\mathbf{x},t) \quad a,b \in \mathbb{R}$$

# Generalisation of the Boundary Conditions Framework

## Boundary Conditions supported in Nektar++ (UserDefined):

4) High Order Pressure Boundary Condition (IncNavierStokesSolver)

$$\frac{\partial p^{n+1}}{\partial n} = -\left[ \frac{\partial \mathbf{u}}{\partial t}^{n+1} + \nu \sum_{q=0}^{J_e-1} \beta_q (\nabla \times \nabla \times \mathbf{u})^{n-q} + \sum_{q-0}^{J_e-1} \beta_q [(\mathbf{u} \cdot \nabla)\mathbf{u}^{n-q} \right] \cdot \mathbf{n}$$

5) Time dependent Boundary conditions

6) Radial Boundary Conditions

7) Axialsymmetric (for cylindical coordinates)

8) …

# Generalisation of the Boundary Conditions Framework

**Library**

**SpatialDomain**

**Conditions.h**

```
enum BoundaryConditionType
{
    eDirchlet,
    eNeumann,
    …
}
enum BndUserDefinedType
{
    eHigh,
    …
}
```

**Conditions.cpp**

```
if(Dirichlet)
{
…
}
else if(Neumann)
{
…
}
```

# Generalisation of the Boundary Conditions Framework

**Library**

**SpatialDomain**

**Conditions.h**

```
DirichletBoundaryCondition(…)
{
...
}

NeumannBoundaryCondition(…)
{
...
}
```

**Conditions.cpp**

```
BoundaryConditionShPtr

bnd=GetBoundaryConditions
Factory().CreateInstance
(…);
```

**Implementation**

**Factory Instance**

Imperial College London

# Generalisation of the Boundary Conditions Framework

```
//Condition.h

// Declaration of the boundary condition factory

typedef LibUtilities::NekFactory<
boost::tuple<std::string,std::string>,
BoundaryConditionBase,
const LibUtilities::SessionReaderSharedPtr&,
const TiXmlElement*> BoundaryConditionsFactory;


struct DirichletBoundaryCondition : public BoundaryConditionBase
{

//Implementation of Dirichlet boundary condition.

…
}
```

# Generalisation of the Boundary Conditions Framework

```
//Condition.cpp

void BoundaryConditions::ReadBoundaryConditions(TiXmlElement
                                                      *conditions)
{
       while(regionElement)
       {
          BoundaryConditionMapShPtr boundaryConditions =
          MemoryManager<BoundaryConditionMap>::AllocateSharedPtr();

          ...
          boost::tuple<std::string,std::string>
          bnd_pair=boost::make_tuple(conditionType,userdefined);

          BoundaryConditionShPtr bnd=GetBoundaryConditionsFactory
          ().CreateInstance(bnd_pair,m_session, conditionElement);

       }
}
```

# Generalisation of the Boundary Conditions Framework

**Advantages of the re-structuring:**

- **Encapsulation of the boundary conditions.**
- **Legibility of the code.**
- **Possibility of implementing new boundary conditions easily.**

Imperial College London

# Conclusions

1. First attempt to reformulate Navier-Stokes equations in cylindrical coordinates
2. Creation of new boundary conditions for the axis.
3. Validation for some test cases (pipe flow, Kovasznay flow)

1. Parallelisation of large scale eigenvalue problems.
2. Possibility of solving eigenproblems for very complex geometries.
3. Good scalability performances.

1. Refactorisation of the boundary condition framework.
2. Encapsulation of the boundary conditions.