GPU TECHNOLOGY CONFERENCE

www.csiro.au

Numerical Simulations in Fluid Dynamics using GPU: a Practical Introduction

Dr Tomasz P Bednarz, Con Caris, Dr John A Taylor

22 September 2010

CSIRO

Content (behind the scene)

- Introduction
  - What is Computational Fluid Dynamics (CFD) and where is it used?
- Governing equations
  - Navier-Stokes equations for conservation of mass, momentum & energy equation for thermal fluid flow
- Discretisation
  - Rectangular and Boundary Fitted Coordinates
- Algorithms
  - HSMAC methodology, UPWIND and UTOPIA schemes.
- Verification
  - Driven Cavity and Natural Convection case.
- Migration to GPU
  - Migration of the existing code to OpenCL.
- Applications
  - Scaling Analysis, Magneto-thermal convection, exchange flows.
- Closing Remarks

CSIRO overview

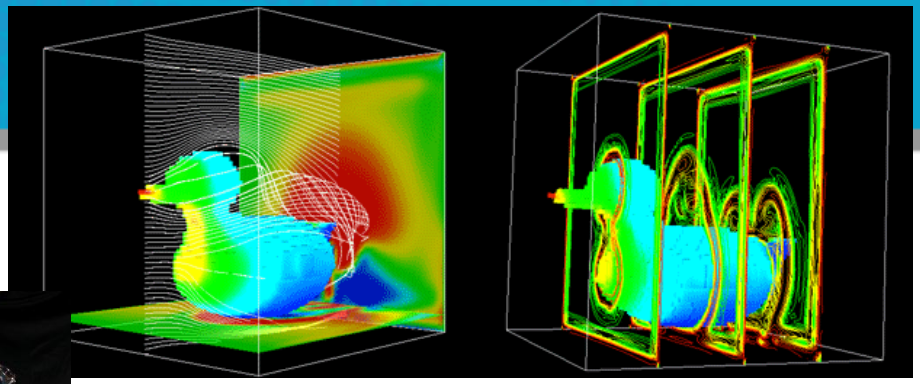
CSIRO today: a snapshot

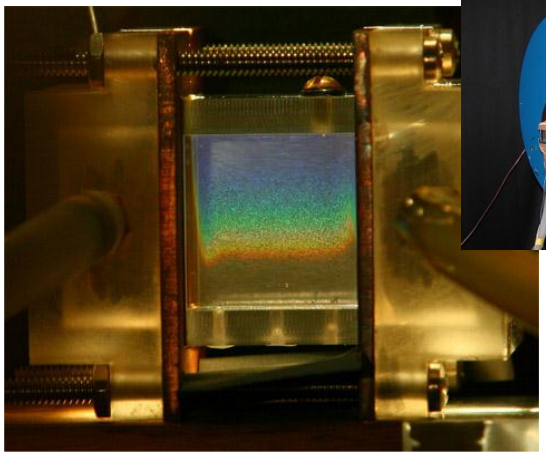
- Australia's national science agency
- One of the largest & most diverse in the world
- 6500+ staff over 55 locations
- Ranked in top 1% in 14 research fields
- 20+ spin-off companies in six years
- 160+ active licences of CSIRO innovation
- Building national prosperity and wellbeing

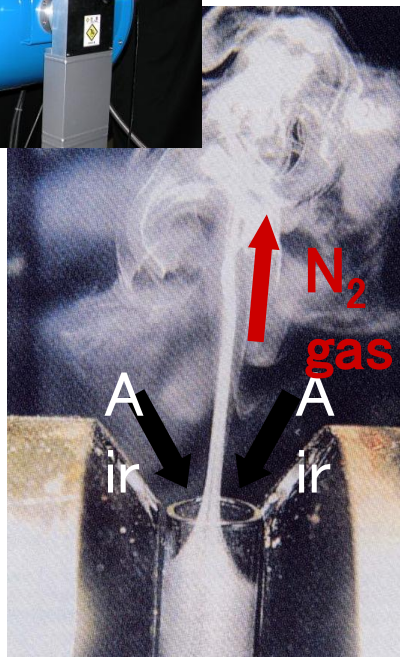
Fluid Dynamics

Introduction

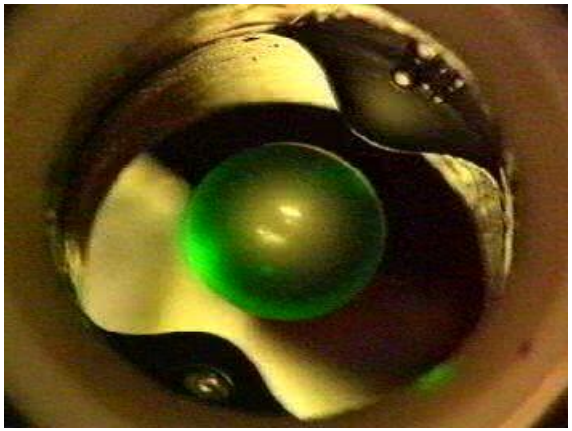
- Experiments and CFD?



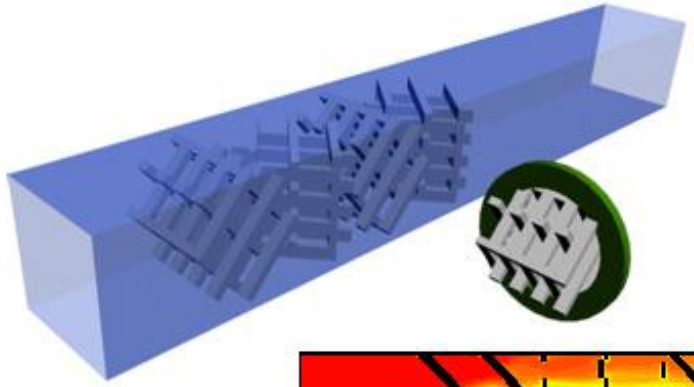




Wakayama Jet

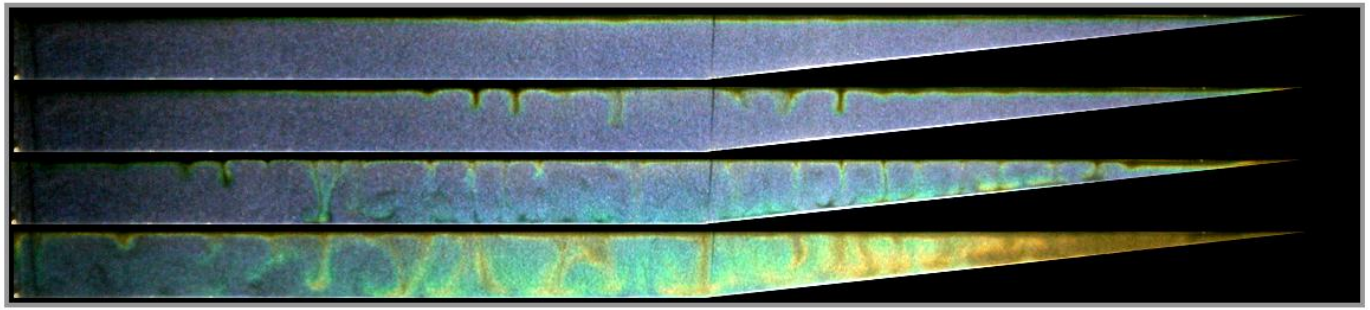


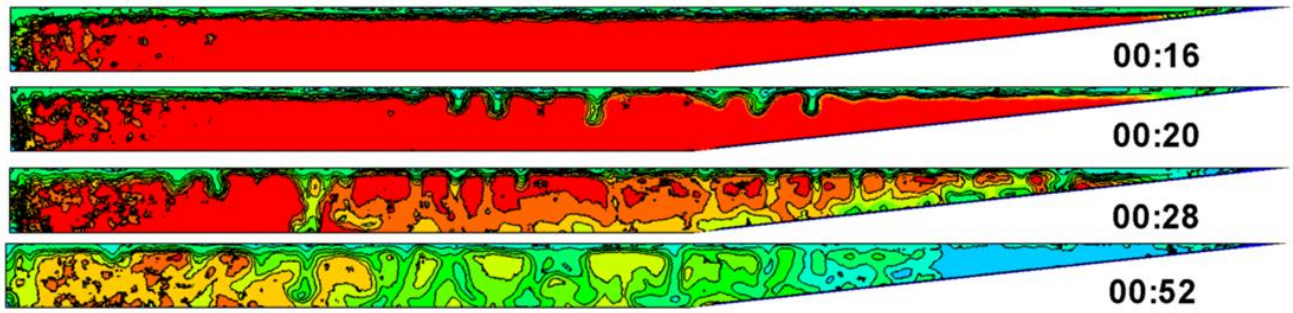
Courtesy of High Field Magnet Laboratory, NL

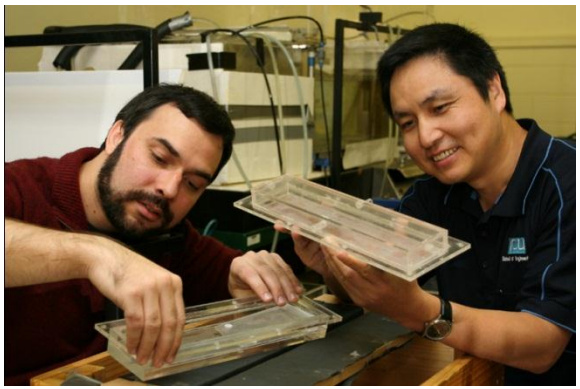


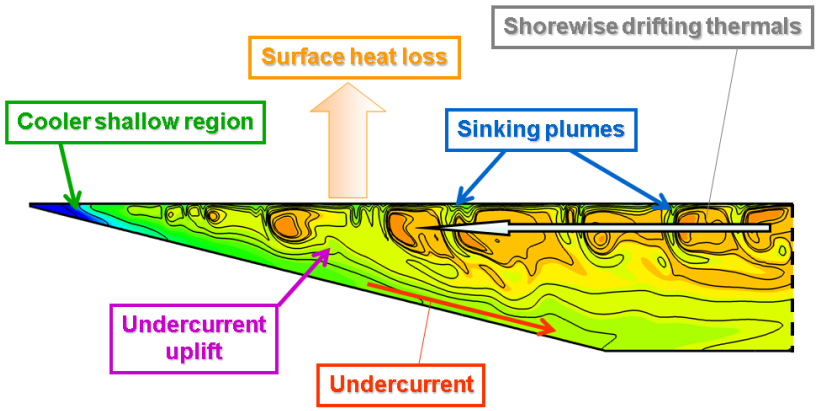
Exchange Flows in Reservoirs – Cooling Case

- Water circulation in reservoirs is driven by thermal gradients changing during day and night cycles.



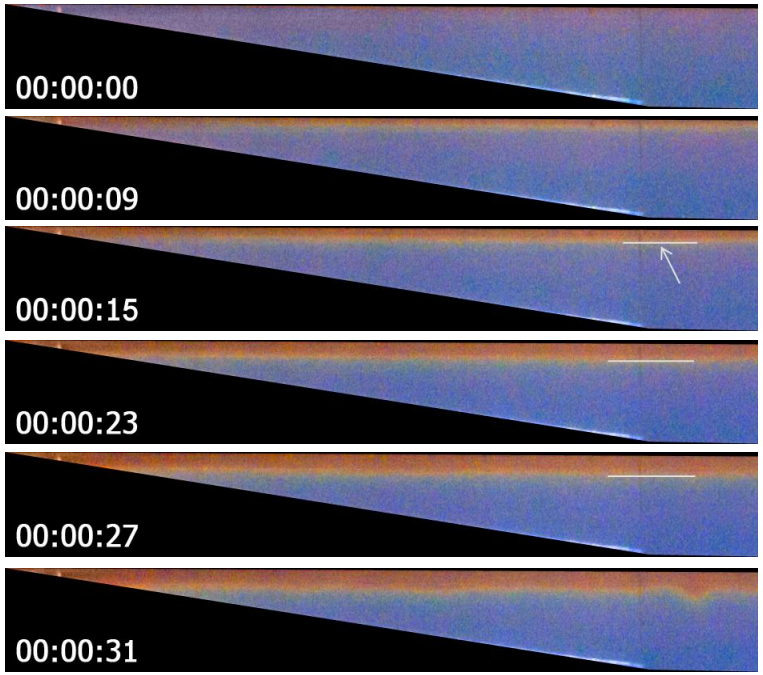




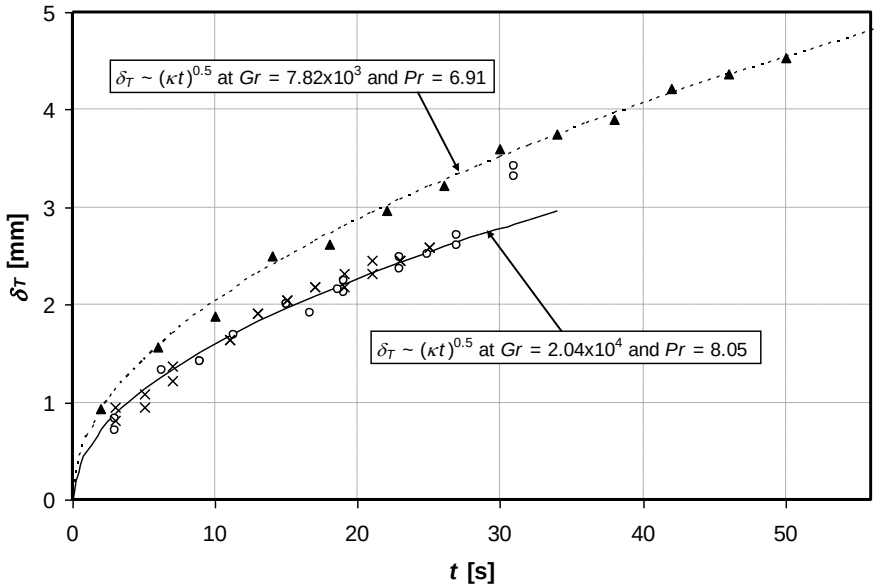


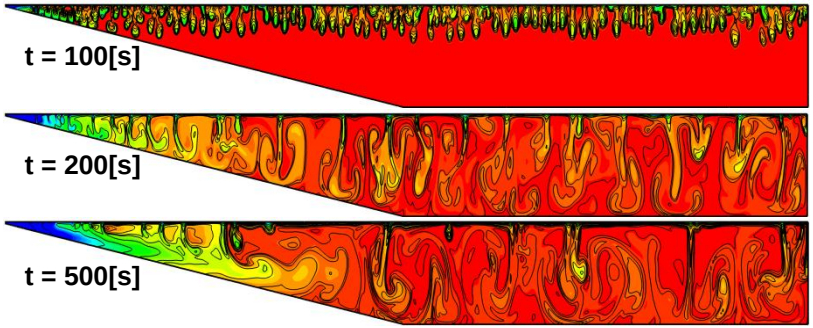
CSIRO

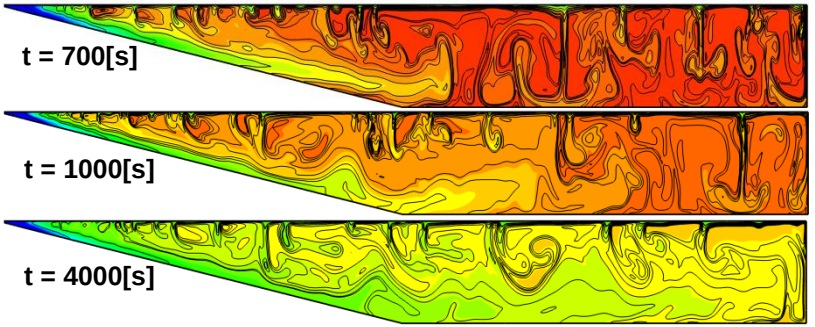
Exchange Flows in Reservoirs – Cooling Case



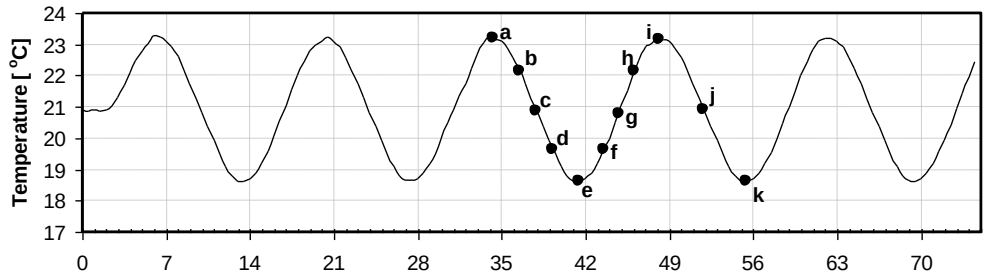
Scaling relation: $\delta_T \sim (\kappa \cdot t)^{1/2}$



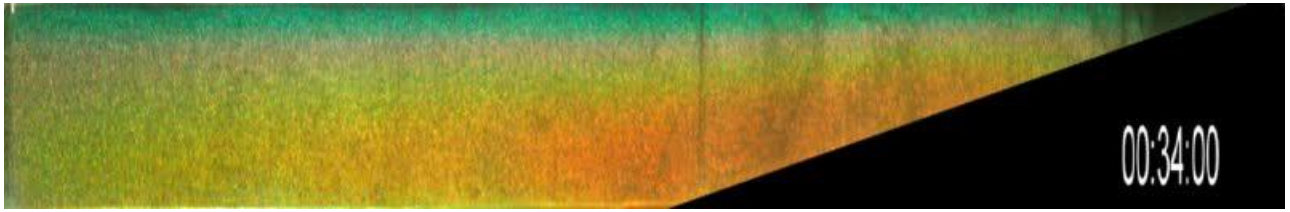


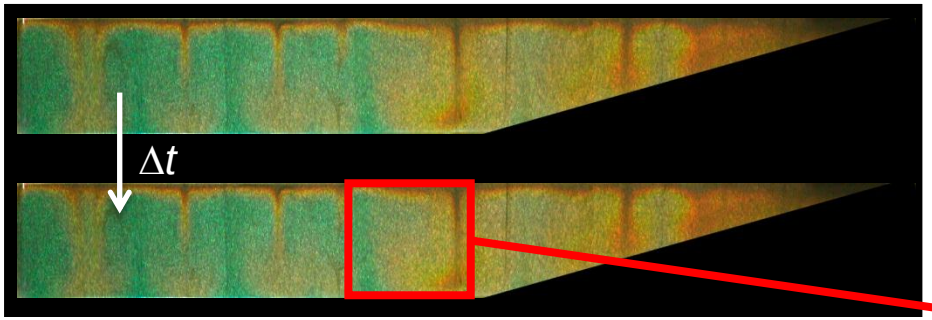


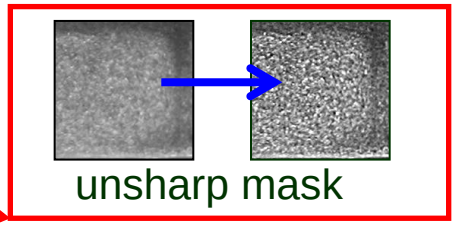
Exchange Flows in Reservoirs – Diurnal Case

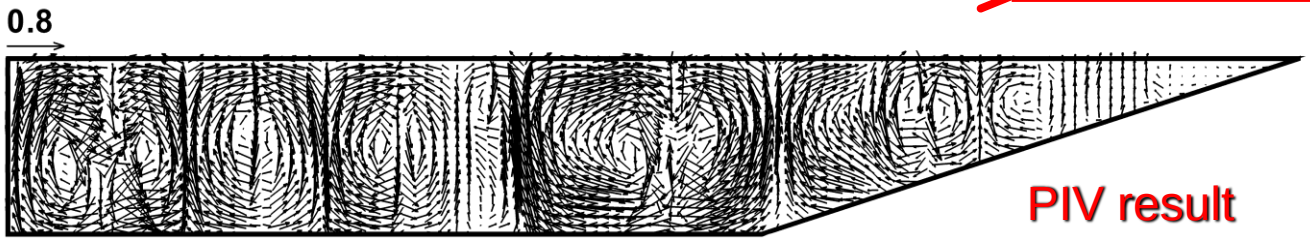


$Pr = 6.82,$
$Gr = 3.52 \times 10^4$



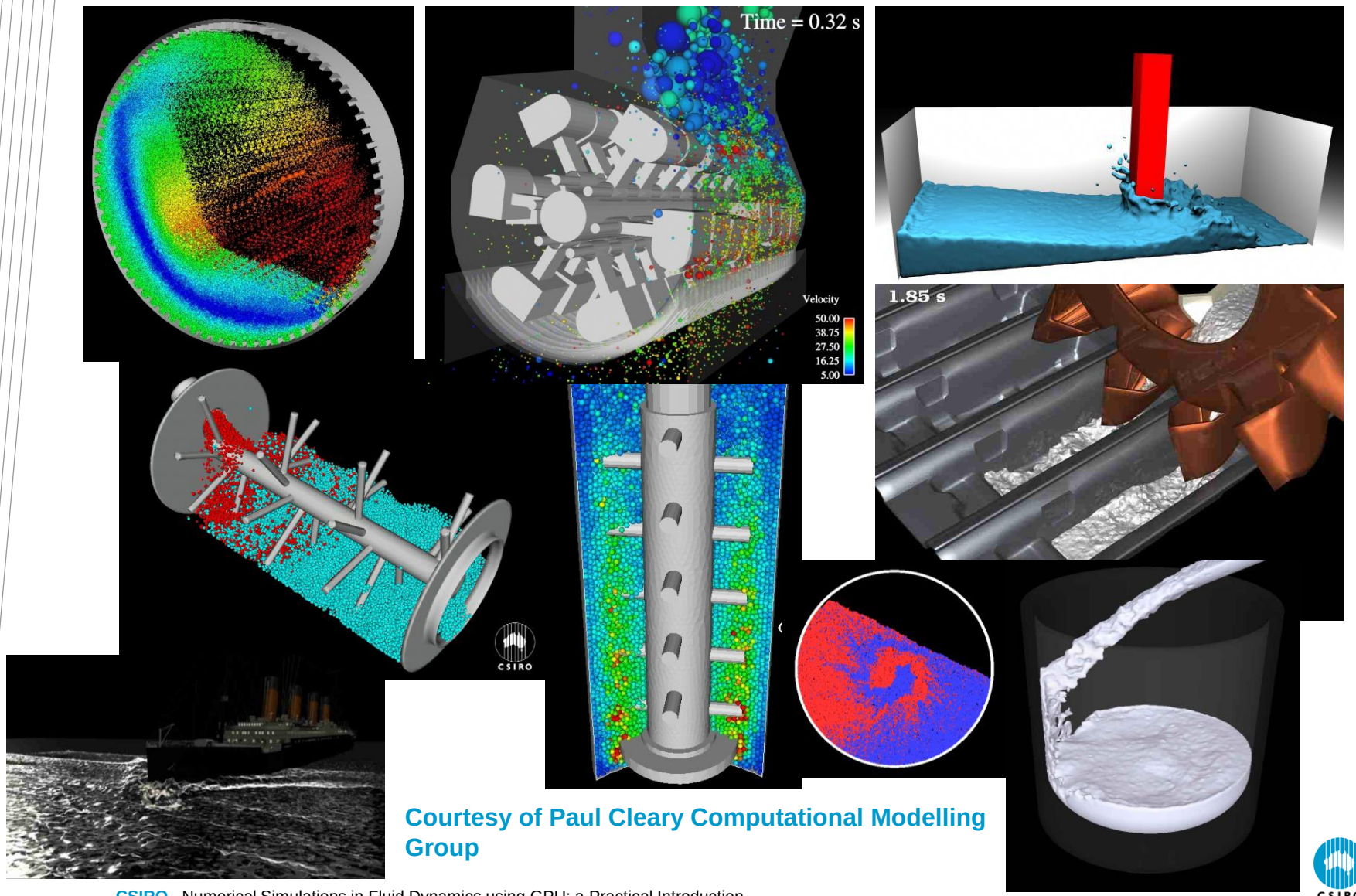






PIV result

DEM and SPH examples



Courtesy of Paul Cleary Computational Modelling Group

Process

Equations → Discretisation → Implementation → Verification → Results → Transfer CPU code to GPU code → Verification → Results!

Governing Equations, Discretisation Procedure, Algorithms.

Governing equations (dimensional)

- Continuity equation

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0$$

- Momentum equations

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+body\_force\_x$$

unsteady acceleration | convective acceleration | pressure gradient | viscous term | body force

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial y}+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)+body\_force\_y$$

- Energy equation

$$\frac{\partial T}{\partial t}+u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}=\alpha\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)$$

Governing equations (non-dimensional)

- Continuity equation

$$\frac{\partial U}{\partial X}+\frac{\partial V}{\partial Y}=0$$

- Momentum equations

$$\frac{\partial U}{\partial \tau}+U\frac{\partial U}{\partial X}+V\frac{\partial U}{\partial Y}=-A\frac{\partial P}{\partial X}+B\left(\frac{\partial^2 U}{\partial X^2}+\frac{\partial^2 U}{\partial Y^2}\right)+CF_X$$

$$\frac{\partial V}{\partial \tau}+U\frac{\partial V}{\partial X}+V\frac{\partial V}{\partial Y}=-A\frac{\partial P}{\partial Y}+B\left(\frac{\partial^2 V}{\partial X^2}+\frac{\partial^2 V}{\partial Y^2}\right)+CF_Y$$

- Energy equation

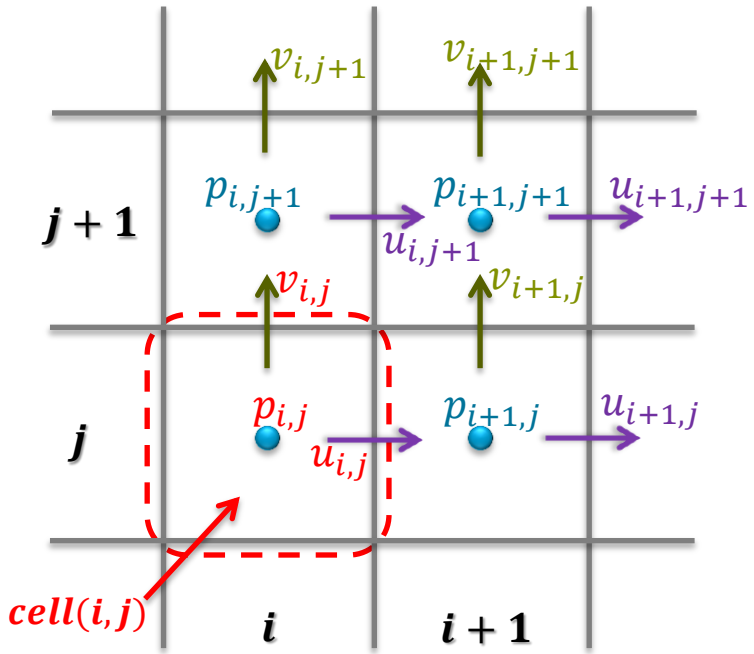
$$\frac{\partial \theta}{\partial \tau}+U\frac{\partial \theta}{\partial X}+V\frac{\partial \theta}{\partial Y}=D\left(\frac{\partial^2 \theta}{\partial X^2}+\frac{\partial^2 \theta}{\partial Y^2}\right)$$

- Where:

$$X=\frac{x}{x^*}\quad Y=\frac{y}{y^*}\quad U=\frac{u}{u^*}\quad V=\frac{v}{v^*}\quad P=\frac{p}{p^*}\quad \tau=\frac{t}{t^*}\quad \theta=\frac{T-T^*}{T^*}$$

Staggered grid

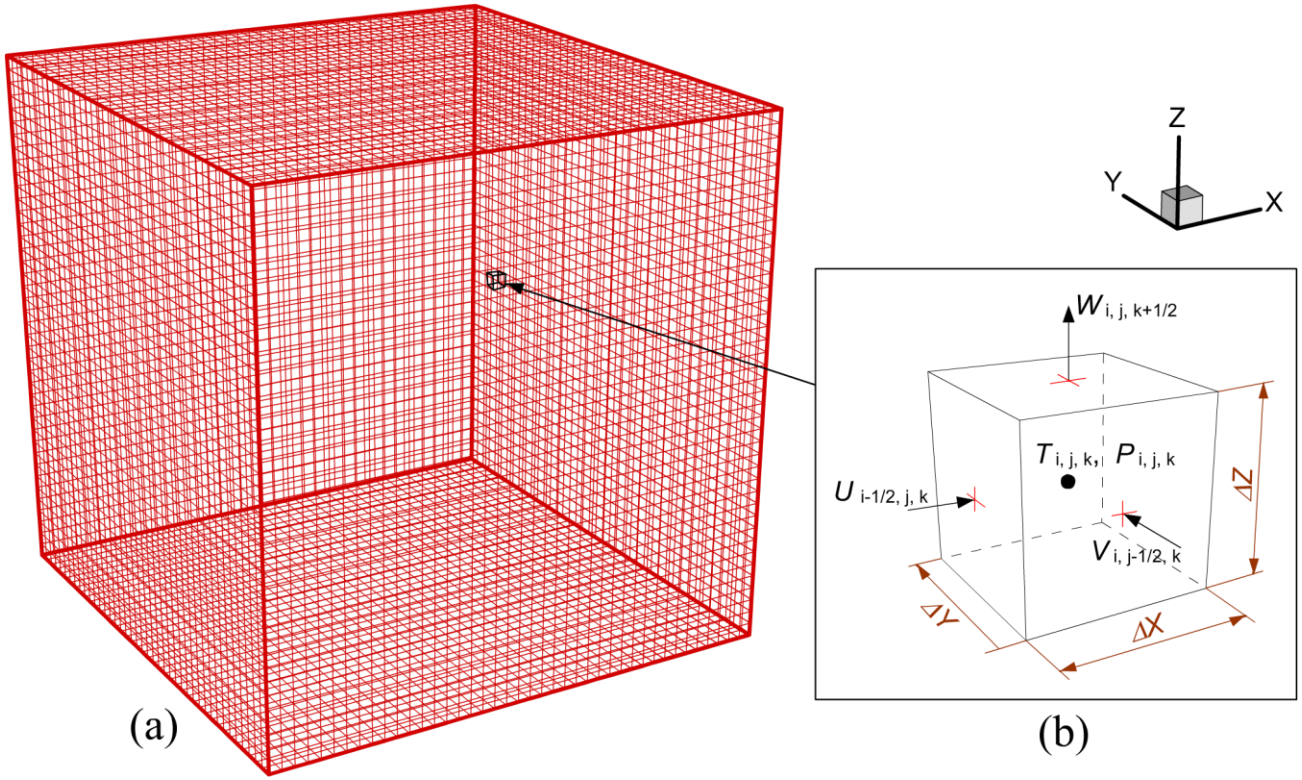
- Different variables are located at different locations:
  - scalar variables (pressure, temperature, concentration) are stored at the centre of the cell,
  - the horizontal velocity component is sampled at the centre of the vertical cell face,
  - the vertical velocity component is sampled at the centre of the horizontal cell face.



- The discrete values of ***u***, ***v*** and ***p*** can be thought as located on three separate grids, each shifted by half grid spacing to the bottom, to the left, and to the lower left respectively.
- The staggered arrangement of the unknowns prevents possible pressure oscillations.

Staggered grid

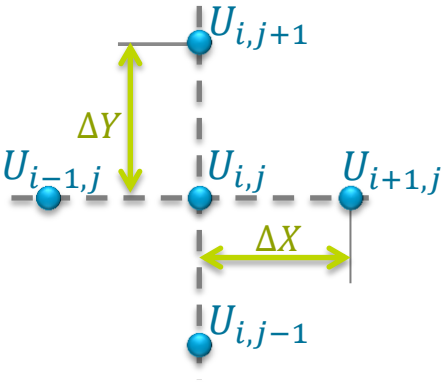
- In 3D, the grid is set up exactly the same way.



Finite Difference (FD) method

- The continuous problem domain is replaced by a finite-difference mesh or grid.
- If we think about $U_{i,j}$ as $U(X_0,Y_0)$ then:

$$U_{i+1,j} = U(X_0 + \Delta X, Y_0) \qquad U_{i-1,j} = U(X_0 - \Delta X, Y_0)$$
$$U_{i,j+1} = U(X_0, Y_0 + \Delta Y) \qquad U_{i,j-1} = U(X_0, Y_0 - \Delta Y)$$



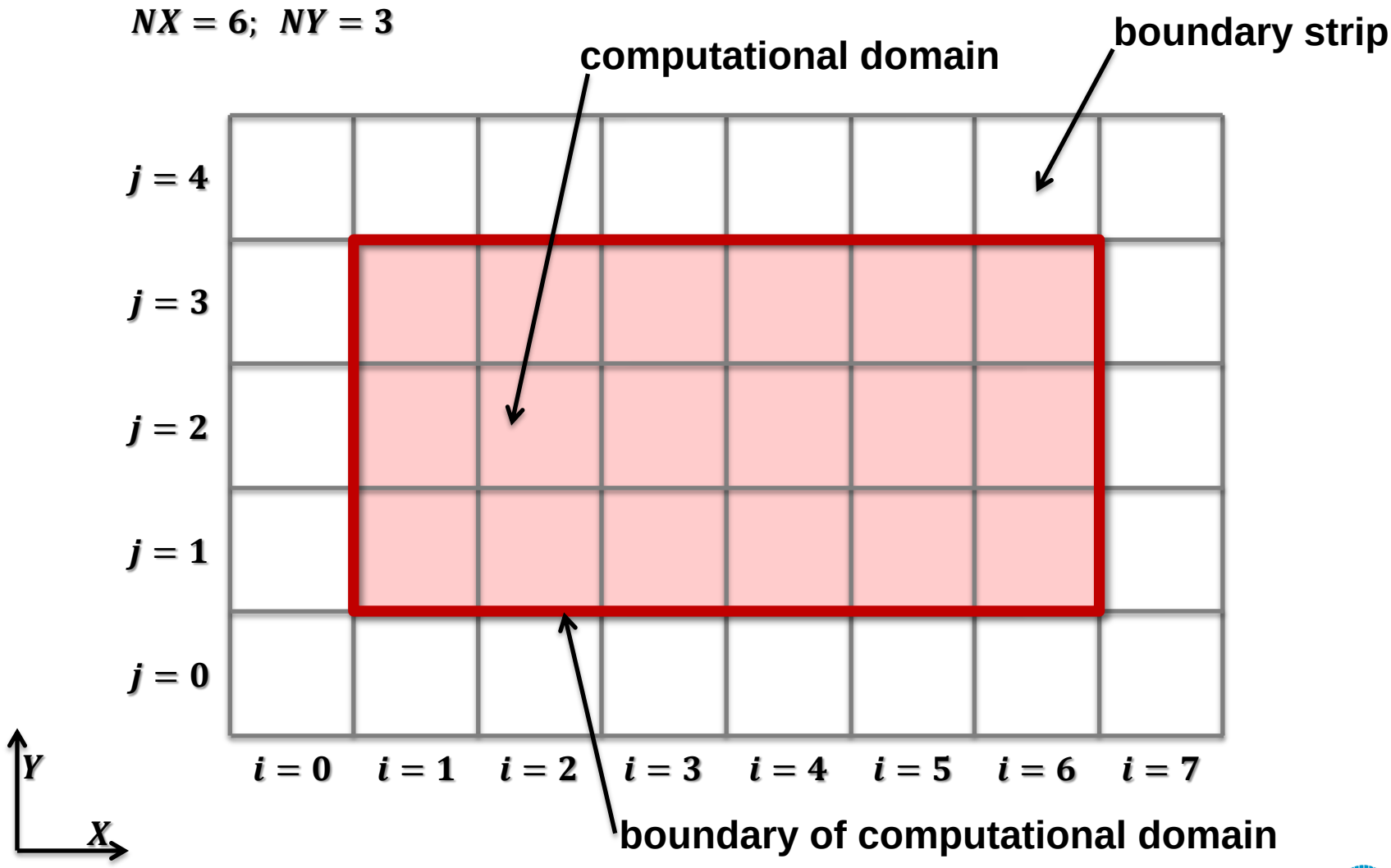
- The idea of FD representation for derivative can be introduced by recalling the definition of the derivative for the function $U(X,Y)$ at X_0 and Y_0:

$$\frac{\partial U}{\partial X} = \lim_{\Delta X \to 0} \frac{U(X_0 + \Delta X, Y_0) - U(X_0, Y_0)}{\Delta X}$$

- The difference approximation can be put on a formal basis through the use of a Taylor-series expansion for $U(X_0+\Delta X,Y_0)$ about (X_0,Y_0):

$$U(X_0 + \Delta X, Y_0) = U(X_0, Y_0) + \left.\frac{\partial U}{\partial X}\right)_0 \Delta X + \left.\frac{\partial^2 U}{\partial X^2}\right)_0 \frac{(\Delta X)^2}{2!} + \left.\frac{\partial^3 U}{\partial X^3}\right)_0 \frac{(\Delta X)^3}{n!} + \cdots$$

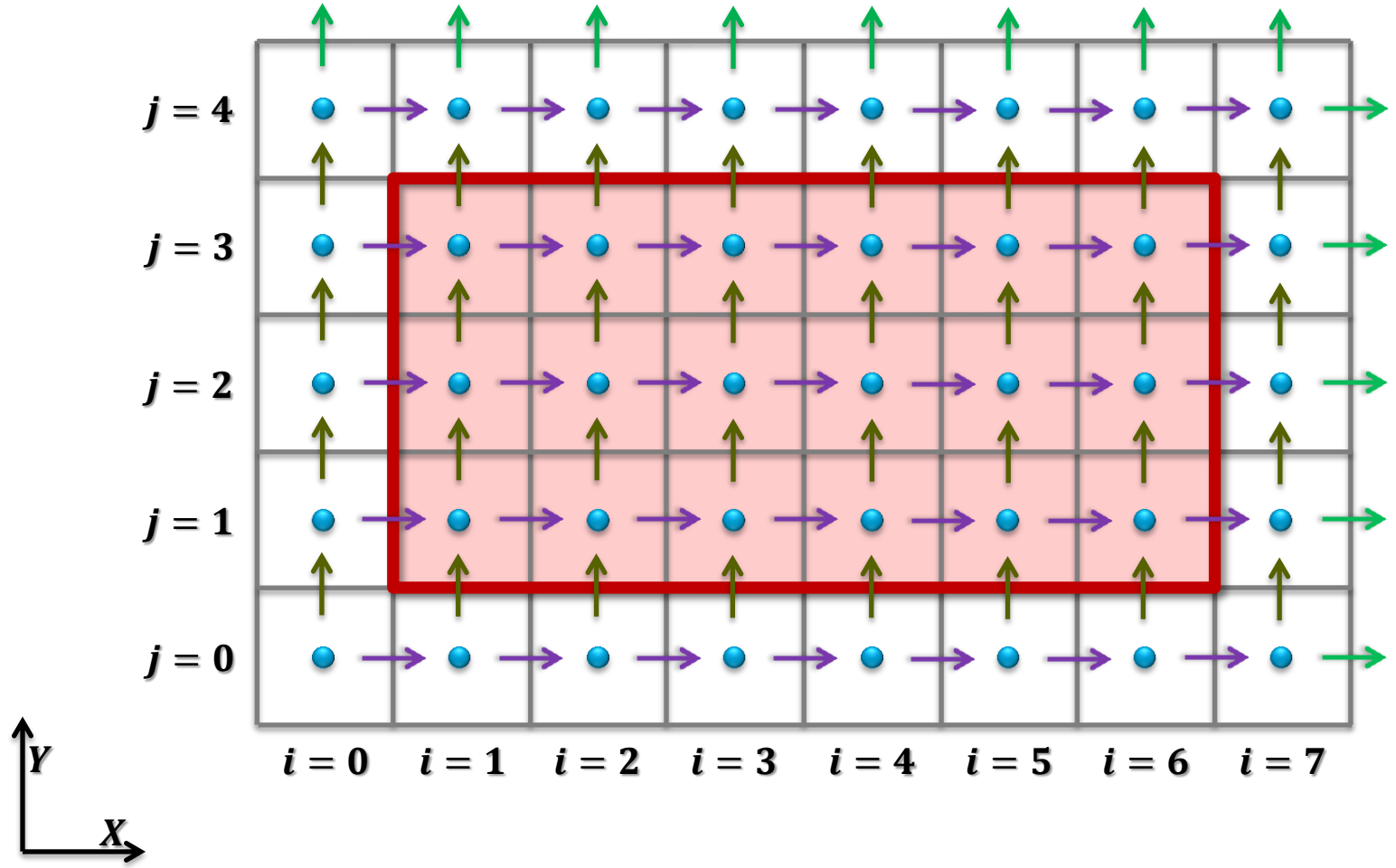
Computational domain



Computational domain

```
// grid nodes
x = std::vector<std::vector<float>>(NX+3, NY+3);
y = std::vector<std::vector<float>>(NX+3, NY+3);
// velocity field
un = std::vector<std::vector<float>>(NX+1, NY+2);
vn = std::vector<std::vector<float>>(NX+2, NY+1);
// pressure field
p = std::vector<std::vector<float>>(NX+2, NY+2);
```

$NX = 6;\ NY = 3$



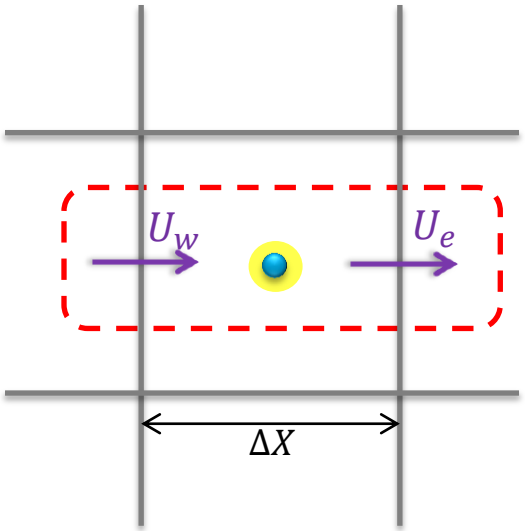
Discretisation of the continuity equation

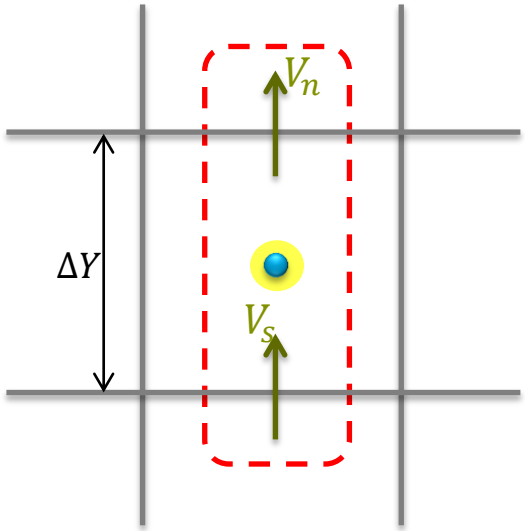
- All derivatives are calculated at the centre of the cell.
- Continuity equation:

$$\frac{\partial U}{\partial X} + \frac{\partial V}{\partial Y} = 0$$

$$U_e = U_{i,j}$$
$$U_w = U_{i-1,j}$$

$$V_n = V_{i,j}$$
$$V_s = V_{i,j-1}$$





$$\frac{\partial U}{\partial X} = \frac{U_e - U_w}{\Delta X}$$

$$\frac{\partial V}{\partial Y} = \frac{V_n - V_s}{\Delta Y}$$

$$\frac{\partial U}{\partial X} + \frac{\partial V}{\partial Y} = \frac{U_{i,j} - U_{i-1,j}}{\Delta X} + \frac{V_{i,j} - V_{i,j-1}}{\Delta Y} = 0$$

```
(un[ i ][ j ]-un[i-1][ j ])/deltaX +
(vn[ i ][ j ]-vn[ i ][j-1])/deltaY
```

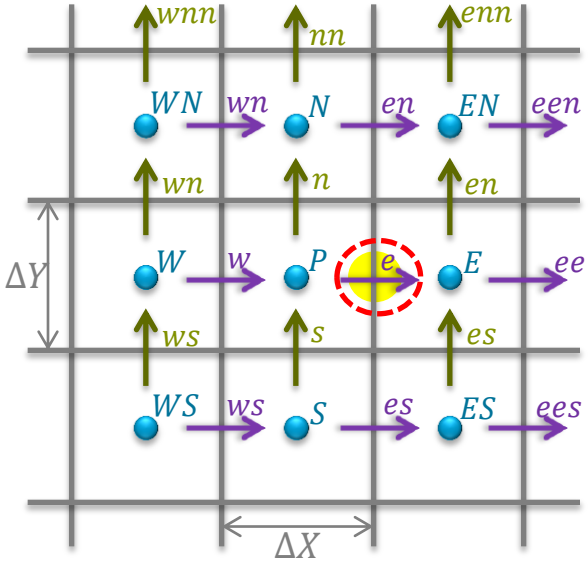
Discretisation of the x-momentum equation

- All derivatives are calculated at the right vertical face of the cell.

$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -A\frac{\partial P}{\partial X} + B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right) + CF_X$$

$$\frac{\partial U}{\partial \tau} = \frac{{U_e}^{k+1} - {U_e}^{k}}{\Delta \tau}$$

- Time advancement.



Discretisation of the x-momentum equation

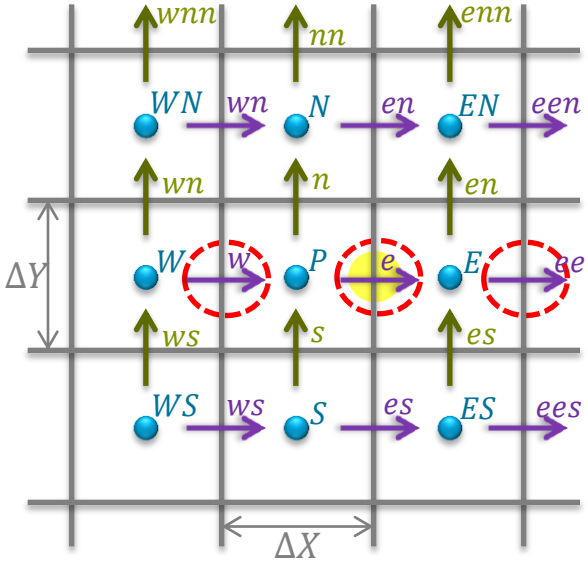
- All derivatives are calculated at the right vertical face of the cell.

$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -A\frac{\partial P}{\partial X} + B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right) + CF_X$$

$$\frac{\partial U}{\partial \tau} = \frac{U_e^{k+1} - U_e^{k}}{\Delta \tau}$$

$$FUX = U\frac{\partial U}{\partial X} = U_e\frac{U_{ee} - U_w}{2\Delta X}$$

```
FUX = uo[i][j]*
    ((uo[i+1][j]-uo[i-1][j])/(2.0f*deltaX))
```



- This term can be improved using upwind scheme !!!

Discretisation of the x-momentum equation

- All derivatives are calculated at the right vertical face of the cell.

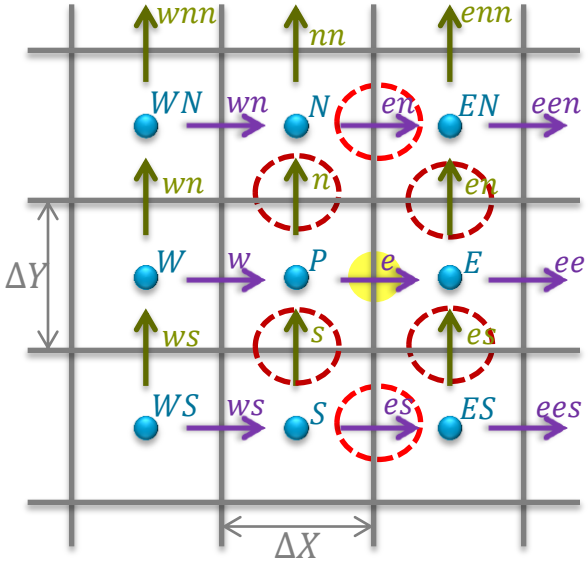
$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -A\frac{\partial P}{\partial X} + B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right) + CF_X$$

$$\frac{\partial U}{\partial \tau} = \frac{U_e{}^{k+1} - U_e{}^{k}}{\Delta \tau}$$

$$FUX = U\frac{\partial U}{\partial X} = U_e\frac{U_{ee} - U_w}{2\Delta X}$$

$$FUY = V\frac{\partial U}{\partial Y} = \left(\frac{V_n + V_{en} + V_{es} + V_s}{4}\right)\frac{U_{en} - U_{es}}{2\Delta Y}$$

```
FUY = ((uo[i][j+1]-uo[i][j-1])/(2.0f*deltaY))*
      ((vo[i][j]+vo[i+1][j]+vo[i][j-1]+vo[i+1][j-1])/4.0f);
```



- This term can be improved using upwind scheme !!!

Discretisation of the x-momentum equation

- All derivatives are calculated at the right vertical face of the cell.

$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -A\frac{\partial P}{\partial X} + B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right) + CF_X$$

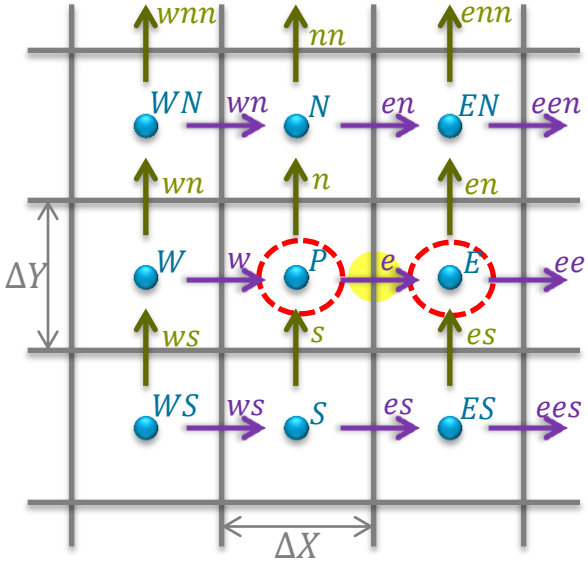
$$\frac{\partial U}{\partial \tau} = \frac{{U_e}^{k+1} - {U_e}^{k}}{\Delta \tau}$$

$$FUX = U\frac{\partial U}{\partial X} = U_e\frac{U_{ee} - U_w}{2\Delta X}$$

$$FUY = V\frac{\partial U}{\partial Y} = \left(\frac{V_n + V_{en} + V_{es} + V_s}{4}\right)\frac{U_{en} - U_{es}}{2\Delta Y}$$

$$PRESSU = A\frac{\partial P}{\partial X} = A\frac{{P_E}^{k+1} - {P_P}^{k+1}}{\Delta X}$$

```
PRESSU = A*(p[i+1][j]-p[i][j])/deltaX;
```



Discretisation of the x-momentum equation

- All derivatives are calculated at the right vertical face of the cell.

$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -A\frac{\partial P}{\partial X} + B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right) + CF_X$$

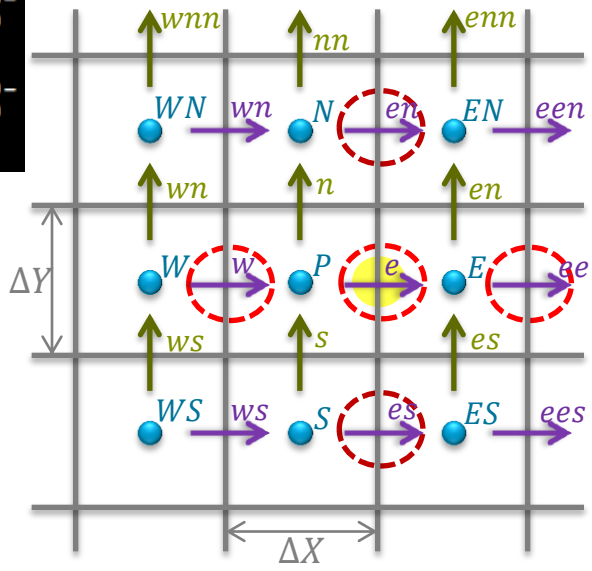
```
udxdx = (
  ((uo[i+1][ j ]-uo[ i ][ j ])/deltaX)-
  ((uo[ i ][ j ]-uo[i-1][ j ])/deltaX)
  )/deltaX;
udydy = (
  ((uo[ i ][j+1]-uo[ i ][ j ])/deltaY)-
  ((uo[ i ][ j ]-uo[ i ][j-1])/deltaY)
  )/deltaY;
DIFFU = B*(udxdx+udydy);
```

$$\frac{\partial U}{\partial \tau} = \frac{{U_e}^{k+1} - {U_e}^{k}}{\Delta \tau}$$

$$FUX = U\frac{\partial U}{\partial X} = U_e\frac{U_{ee} - U_w}{2\Delta X}$$

$$FUY = V\frac{\partial U}{\partial Y} = \left(\frac{V_n + V_{en} + V_{es} + V_s}{4}\right)\frac{U_{en} - U_{es}}{2\Delta Y}$$

$$PRESSU = A\frac{\partial P}{\partial X} = A\frac{{P_E}^{k+1} - {P_P}^{k+1}}{\Delta X}$$



$$DIFFU = B\left\{\left(\frac{U_{ee} - U_e}{\Delta X} - \frac{U_e - U_w}{\Delta X}\right)\frac{1}{\Delta X} + \left(\frac{U_{en} - U_e}{\Delta Y} - \frac{U_e - U_{es}}{\Delta Y}\right)\frac{1}{\Delta Y}\right\}$$

Discretisation of the y-momentum equation

- All derivatives are calculated at the upper horizontal face of the cell.

$$\frac{\partial V}{\partial \tau} + U\frac{\partial V}{\partial X} + V\frac{\partial V}{\partial Y} = -A\frac{\partial P}{\partial Y} + B\left(\frac{\partial^2 V}{\partial X^2} + \frac{\partial^2 V}{\partial Y^2}\right) + CF_Y$$

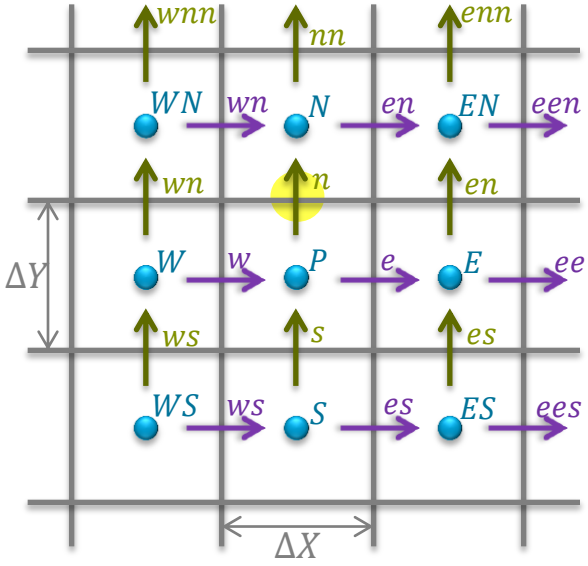
$$\frac{\partial V}{\partial \tau} = \frac{{V_n}^{k+1} - {V_n}^{k}}{\Delta \tau}$$

$$FVX = U\frac{\partial V}{\partial X} = \left(\frac{U_{wn} + U_{en} + U_e + U_w}{4}\right)\frac{V_{en} - V_{wn}}{2\Delta X}$$

$$FVY = V\frac{\partial V}{\partial Y} = V_n\frac{V_{nn} - V_s}{2\Delta Y}$$

$$PRESSV = A\frac{\partial P}{\partial Y} = A\frac{{P_N}^{k+1} - {P_P}^{k+1}}{\Delta Y}$$

$$DIFFV = B\left\{\left(\frac{V_{en} - V_n}{\Delta X} - \frac{V_n - V_{wn}}{\Delta X}\right)\frac{1}{\Delta X} + \left(\frac{V_{nn} - V_n}{\Delta Y} - \frac{V_n - V_s}{\Delta Y}\right)\frac{1}{\Delta Y}\right\}$$



Discretisation of the energy equation

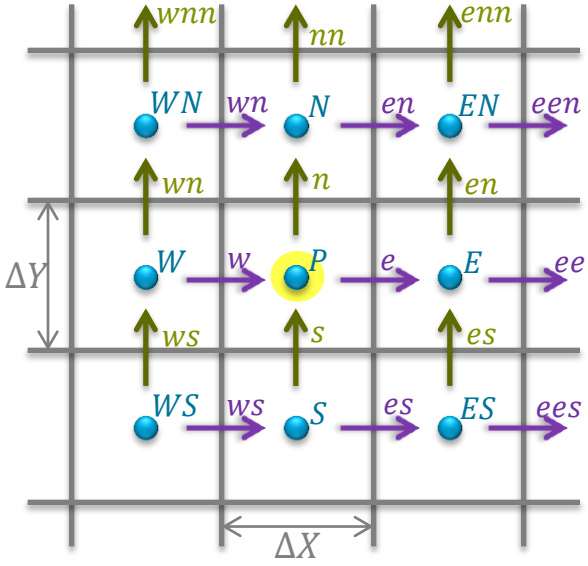
- All derivatives are calculated in the centre of the cell.

$$\frac{\partial\theta}{\partial\tau} + U\frac{\partial\theta}{\partial X} + V\frac{\partial\theta}{\partial Y} = D\left(\frac{\partial^2\theta}{\partial X^2} + \frac{\partial^2\theta}{\partial Y^2}\right)$$

$$\frac{\partial\theta}{\partial\tau} = \frac{{\theta_P}^{k+1} - {\theta_P}^{k}}{\Delta\tau}$$

$$FTX = U\frac{\partial\theta}{\partial X} = \left(\frac{U_e + U_w}{2}\right)\frac{\theta_E - \theta_W}{2\Delta X}$$

$$FTY = V\frac{\partial\theta}{\partial Y} = \left(\frac{V_n + V_s}{2}\right)\frac{\theta_N - \theta_S}{2\Delta Y}$$



$$DIFFT = D\left\{\left(\frac{\theta_E - \theta_P}{\Delta X} - \frac{\theta_P - \theta_W}{\Delta X}\right)\frac{1}{\Delta X} + \left(\frac{\theta_N - \theta_P}{\Delta Y} - \frac{\theta_P - \theta_S}{\Delta Y}\right)\frac{1}{\Delta Y}\right\}$$

Velocity iteration

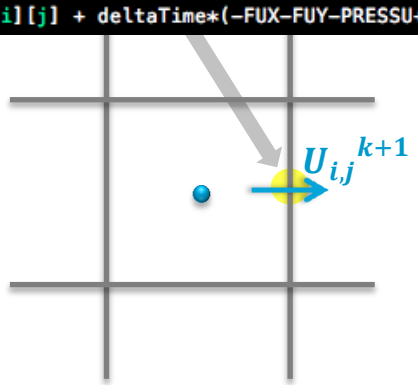
$$\frac{\partial U}{\partial \tau} + \underbrace{U\frac{\partial U}{\partial X}}_{FUX} + \underbrace{V\frac{\partial U}{\partial Y}}_{FUY} = \underbrace{-A\frac{\partial P}{\partial X}}_{PRESSU} + \underbrace{B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right)}_{DIFFU} + \underbrace{CF_X}_{FX}$$

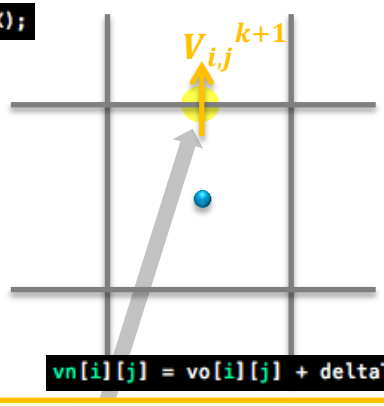
$$\frac{\partial V}{\partial \tau} + \underbrace{U\frac{\partial V}{\partial X}}_{FVX} + \underbrace{V\frac{\partial V}{\partial Y}}_{FVY} = \underbrace{-A\frac{\partial P}{\partial Y}}_{PRESSV} + \underbrace{B\left(\frac{\partial^2 V}{\partial X^2} + \frac{\partial^2 V}{\partial Y^2}\right)}_{DIFFV} + \underbrace{CF_Y}_{FY}$$

- Momentum equations are rewritten to get velocities:

$$U_{i,j}^{k+1} = U_{i,j}^{k} + \Delta\tau(-FUX - FUY - PRESSU + DIFFU + FX)$$

```
un[i][j] = uo[i][j] + deltaTime*(-FUX-FUY-PRESSU+DIFFU+FX);
```






```
vn[i][j] = vo[i][j] + deltaTime * (-FVX-FVY-PRESSV+DIFFV+FY);
```

$$V_{i,j}^{k+1} = V_{i,j}^{k} + \Delta\tau(-FVX - FVY - PRESSV + DIFFV + FY)$$

Temperature iteration

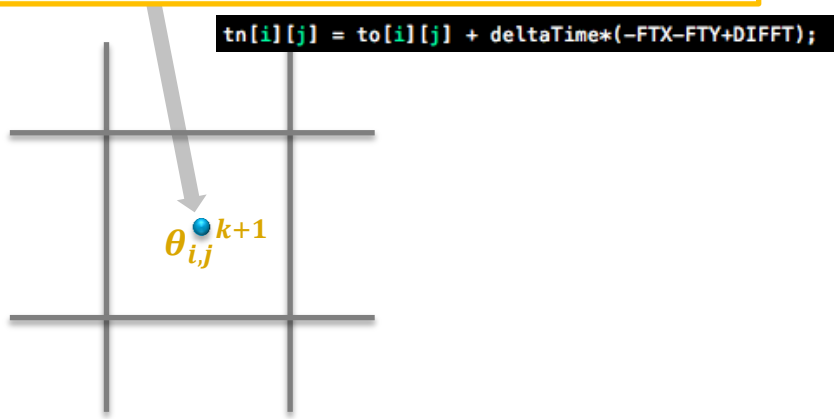
- Energy equation:

$$\frac{\partial \theta}{\partial \tau} + \underbrace{U\frac{\partial \theta}{\partial X}}_{FTX} + \underbrace{V\frac{\partial \theta}{\partial Y}}_{FTY} = \underbrace{D\left(\frac{\partial^2 \theta}{\partial X^2} + \frac{\partial^2 \theta}{\partial Y^2}\right)}_{DIFFT}$$

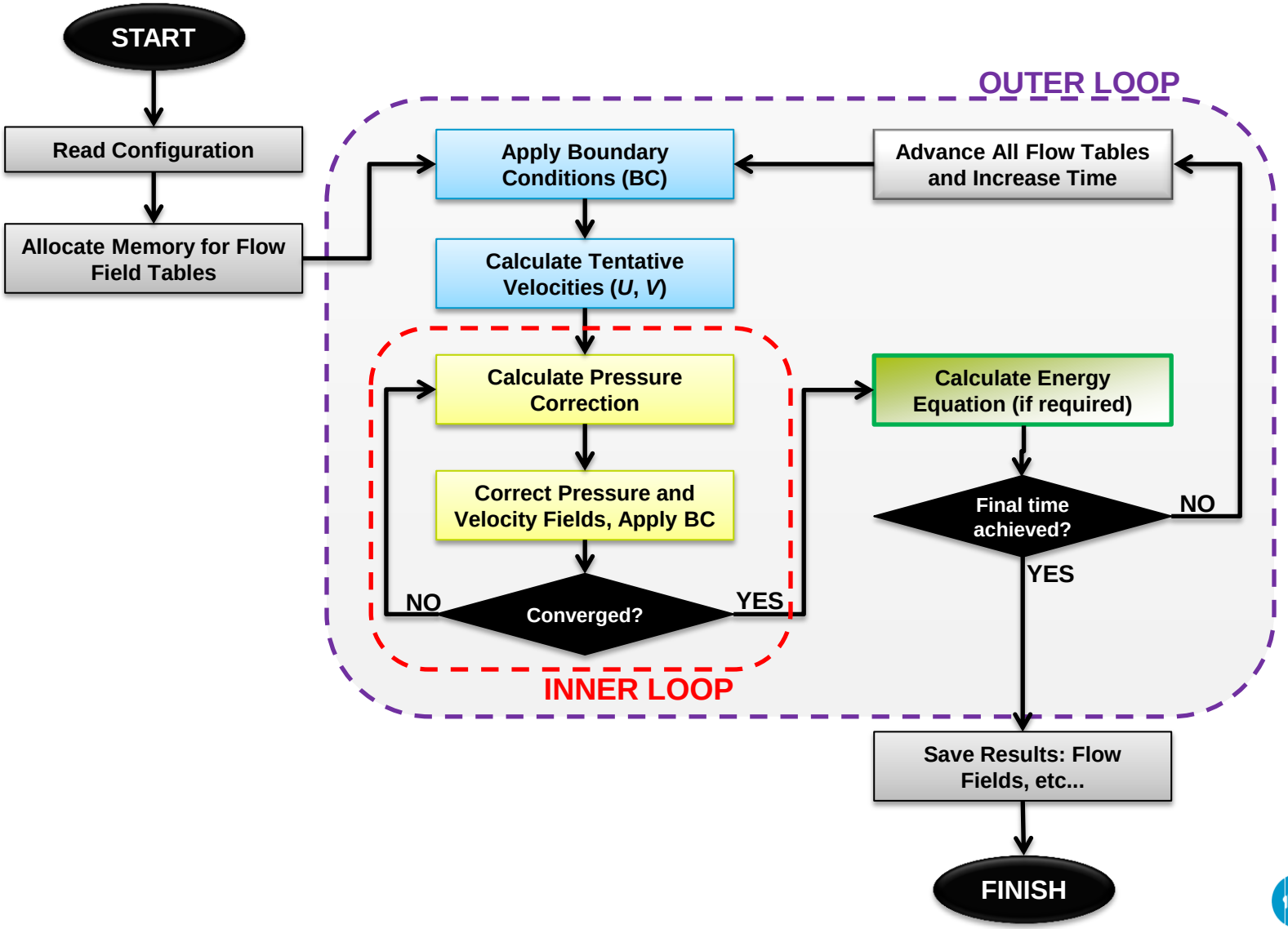
- Energy equation is rewritten to get temperature field:

$$\theta_{i,j}{}^{k+1} = \theta_{i,j}{}^{k} + \Delta\tau(-FTX - FTY + DIFFU)$$

```
tn[i][j] = to[i][j] + deltaTime*(-FTX-FTY+DIFFT);
```



Flow Chart of the CPU Fluid Solver



HSMAC

Amsden and Welch, JPC 10, 324-325, 1970.
Harlow and Welch, Physics of Fluids 8, 2182-2189, 1965.

- HSMAC (**H**ighly **S**implified **M**arker **a**nd **C**ell): a method to solve for pressure and velocity components.
- Momentum equations:

$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -A\frac{\partial P}{\partial X} + B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right) + CF_X$$

$$\frac{\partial V}{\partial \tau} + U\frac{\partial V}{\partial X} + V\frac{\partial V}{\partial Y} = -A\frac{\partial P}{\partial Y} + B\left(\frac{\partial^2 V}{\partial X^2} + \frac{\partial^2 V}{\partial Y^2}\right) + CF_Y$$

- Can be rewritten as follows:

$$U_{i,j}{}^{k+1} = U_{i,j}{}^{k} + \Delta\tau\left(F_{i,j}{}^{k} - A\frac{P_{i+1,j}{}^{k+1} - P_{i,j}{}^{k+1}}{\Delta X}\right)$$

$$V_{i,j}{}^{k+1} = V_{i,j}{}^{k} + \Delta\tau\left(G_{i,j}{}^{k} - A\frac{P_{i,j+1}{}^{k+1} - P_{i,j}{}^{k+1}}{\Delta Y}\right)$$

- Once new pressure are given, we can get new velocity components from these equations.

HSMAC

- These velocity components are required to satisfy the equation of continuity:

$$\frac{\partial U}{\partial X} + \frac{\partial V}{\partial Y} = 0$$

- The solution is sought with the Newton-Raphson method in stepwise way:
  - At first, with the tentative value for **pressure**, the tentative **velocity** components are computed:

$${}^{m}U_{i,j}{}^{k+1} = U_{i,j}{}^{k} + \Delta\tau \left(F_{i,j}{}^{k} - A \frac{{}^{m}P_{i+1,j}{}^{k+1} - {}^{m}P_{i,j}{}^{k+1}}{\Delta X} \right)$$

$${}^{m}V_{i,j}{}^{k+1} = V_{i,j}{}^{k} + \Delta\tau \left(G_{i,j}{}^{k} - A \frac{{}^{m}P_{i,j+1}{}^{k+1} - {}^{m}P_{i,j}{}^{k+1}}{\Delta Y} \right)$$

HSMAC

- The tentative velocity and the pressure fields need to be corrected to give better approximation at step m+1:

$$
\begin{aligned}
{}^{m+1}U^{k+1} &= {}^{m}U^{k+1} + U' \\
{}^{m+1}V^{k+1} &= {}^{m}V^{k+1} + V' \\
{}^{m+1}P^{k+1} &= {}^{m}P^{k+1} + P'
\end{aligned}
$$

- Then, the LHS of continuity equation is expressed as:

$$
{}^{m+1}D_{i,j}^{k+1} = \frac{{}^{m+1}U_{i,j}^{k+1} - {}^{m+1}U_{i-1,j}^{k+1}}{\Delta X} + \frac{{}^{m+1}V_{i,j}^{k+1} - {}^{m+1}V_{i,j-1}^{k+1}}{\Delta Y}
$$

$$
{}^{m}D_{i,j}^{k+1} = \frac{{}^{m}U_{i,j}^{k+1} - {}^{m}U_{i-1,j}^{k+1}}{\Delta X} + \frac{{}^{m}V_{i,j}^{k+1} - {}^{m}V_{i,j-1}^{k+1}}{\Delta Y}
$$

- The equation of continuity is to be satisfied in the new time step $(k+1)\Delta\tau$ so let us assume $D_{i,j}^{k+1} = 0$ though $D_{i,j}^{k} \neq 0$ with the non-convergent incorrect velocity components for U and V.

- Newton-Raphson method provides the stepwise approach for the solution to satisfy $D_{i,j}^{k+1} = 0$.

HSMAC

$$ {}^{m}U_{i,j}{}^{k+1} = U_{i,j}{}^{k} + \Delta\tau\left(F_{i,j}{}^{k} - \frac{{}^{m}P_{i+1,j}{}^{k+1} - {}^{m}P_{i,j}{}^{k+1}}{\Delta X}\right) $$

$$ {}^{m}V_{i,j}{}^{k+1} = V_{i,j}{}^{k} + \Delta\tau\left(G_{i,j}{}^{k} - \frac{{}^{m}P_{i,j+1}{}^{k+1} - {}^{m}P_{i,j}{}^{k+1}}{\Delta Y}\right) $$

- These are inserted in continuity equation:

$$ \frac{U_{i,j}^{k+1} - U_{i-1,j}^{k+1}}{\Delta X} + \frac{V_{i,j}^{k+1} - V_{i,j-1}^{k+1}}{\Delta Y} = D_{i,j}^{k+1} $$

$$ \frac{U_{i,j}^{k} - U_{i-1,j}^{k}}{\Delta X} + \frac{V_{i,j}^{k} - V_{i,j-1}^{k}}{\Delta Y} = D_{i,j}^{k} = 0 $$

$$ + \Delta\tau\left\{\frac{F_{i,j}^{k} - F_{i-1,j}^{k}}{\Delta X} + \frac{G_{i,j}^{k} - G_{i,j-1}^{k}}{\Delta Y}\right. $$

$$ + \left[-\frac{P_{i+1,j}^{k+1} - P_{i,j}^{k+1}}{\Delta X} + \frac{P_{i,j}^{k+1} - P_{i-1,j}^{k+1}}{\Delta X}\right]\frac{1}{\Delta X} $$

$$ \left. + \left[-\frac{P_{i,j+1}^{k+1} - P_{i,j}^{k+1}}{\Delta Y} + \frac{P_{i,j}^{k+1} - P_{i,j-1}^{k+1}}{\Delta Y}\right]\frac{1}{\Delta Y}\right\} $$

HSMAC

- Calculate P_P so D_P^{k+1} becomes zero using Newton-Raphson method:

$$ {}^{m+1}P_{i,j}^{k+1} = {}^{m}P_{i,j}^{k+1} - \frac{{}^{m}D_{i,j}^{k+1}}{{}^{m}\left(\frac{\partial D_{i,j}}{\partial P_{i,j}}\right)^{k+1}} = {}^{m}P_{i,j}^{k+1} + P'_{i,j} $$

- where:

$$ {}^{m}D_{i,j}^{k+1} = \frac{{}^{m}U_{i,j}^{k+1} - {}^{m}U_{i-1,j}^{k+1}}{\Delta X} + \frac{{}^{m}V_{i,j}^{k+1} - {}^{m}V_{i,j-1}^{k+1}}{\Delta Y} = DD $$

$$ {}^{m}\left(\frac{\partial D_{i,j}}{\partial P_{i,j}}\right)^{k+1} = \Delta\tau\left\{\frac{1}{\Delta X}\left(\frac{1}{\Delta X}+\frac{1}{\Delta X}\right)+\frac{1}{\Delta Y}\left(\frac{1}{\Delta Y}+\frac{1}{\Delta Y}\right)\right\} = DEL $$

- so:

$$ P'_{i,j} = -\frac{DD}{DEL} $$

HSMAC

- Since the pressure is corrected, the velocity components should be corrected to reflect this correction in pressure.
- Taylor expansion of velocity components with pressure in terms of small pressure increment P'_P:

$$ {}^{m+1}U_{i,j}^{k+1} = {}^{m}U_{i,j}^{k+1} + {}^{m}\left(\frac{\partial U_{i,j}}{\partial P_{i,j}}\right)^{k+1} P'_{i,j} + \cdots = {}^{m}U_{i,j}^{k+1} + U'_{i,j} + \cdots $$

$$ {}^{m+1}V_{i,j}^{k+1} = {}^{m}V_{i,j}^{k+1} + {}^{m}\left(\frac{\partial V_{i,j}}{\partial P_{i,j}}\right)^{k+1} P'_{i,j} + \cdots = {}^{m}V_{i,j}^{k+1} + V'_{i,j} + \cdots $$

$$ U'_e = \frac{\Delta\tau}{\Delta X} P'_{i,j} \qquad V'_n = \frac{\Delta\tau}{\Delta Y} P'_{i,j} $$

$$ U'_w = -\frac{\Delta\tau}{\Delta X} P'_{i,j} \qquad V'_s = -\frac{\Delta\tau}{\Delta Y} P'_{i,j} $$

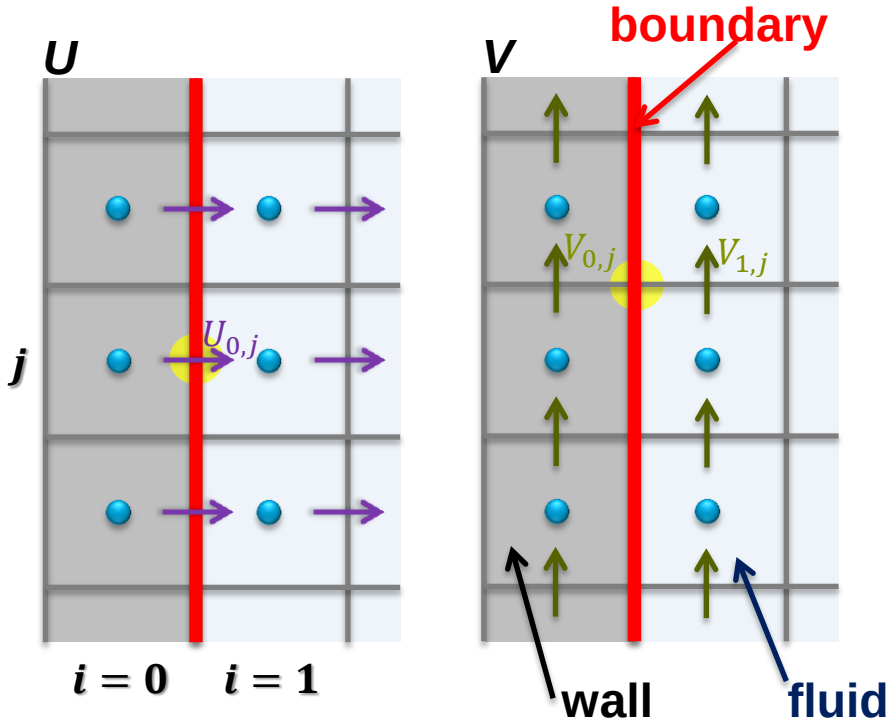
Non-slip boundary conditions

- To satisfy the non-slip condition, the continuous velocities should vanish at the boundary:

$$U_{boundary} = 0; \quad V_{boundary} = 0$$

- Therefore:

$$U_{0,j} = 0; \quad V_{0,j} = -V_{1,j}$$




```
unsigned int j;

// LEFT BOUNDARY
// horizontal velocity component
for (j=1; j<=NY; j++)
{
  un[0][j] = 0.0f;
}
// vertical velocity component
for (j=0; j<=NY; j++)
{
  vn[0][j] = -vn[1][j];
}

/*
where:
std::vector<std::vector<float>> un;
std::vector<std::vector<float>> vn;

un = std::vector<std::vector<float>>(NX+1, NY+2);
vn = std::vector<std::vector<float>>(NX+2, NY+1);
*/
```

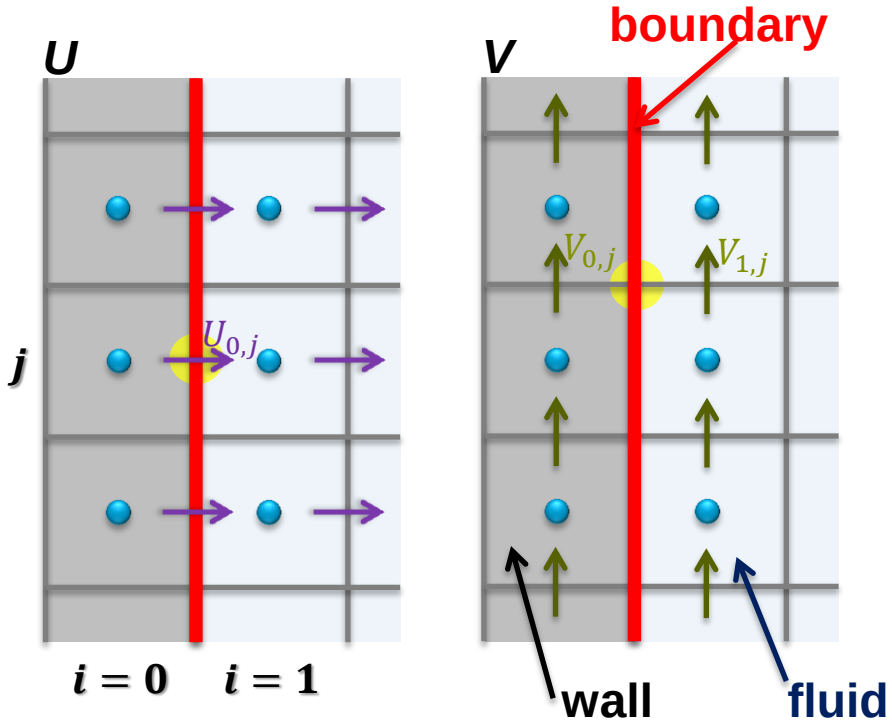
Free-slip boundary conditions

- The velocity component normal to boundary and normal derivative of velocity component tangent to boundary should vanish:

$$U_{boundary} = 0; \quad \partial V / \partial n_{boundary} = 0$$

- Therefore:

$$U_{0,j} = 0; \quad V_{0,j} = V_{1,j}$$




```
unsigned int j;

// LEFT BOUNDARY
// horizontal velocity component
for (j=1; j<=NY; j++)
{
  un[0][j] = 0.0f;
}
// vertical velocity component
for (j=0; j<=NY; j++)
{
  vn[0][j] = vn[1][j];
}

/*
where:
std::vector<std::vector<float>> un;
std::vector<std::vector<float>> vn;

un = std::vector<std::vector<float>>(NX+1, NY+2);
vn = std::vector<std::vector<float>>(NX+2, NY+1);
*/
```

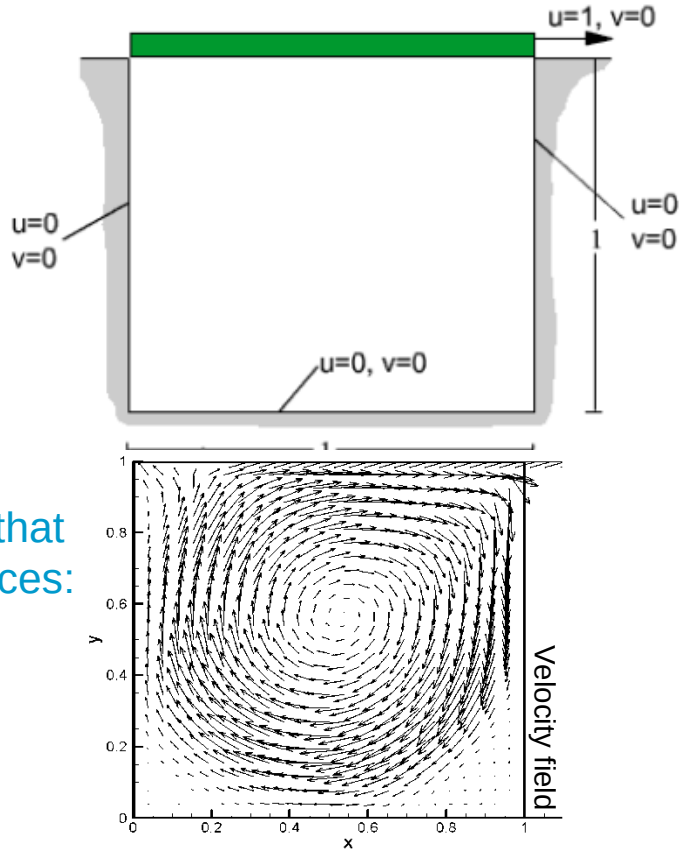
Case 1: Lid Driven Cavity

- Used as a validation case for new codes (same here?).
- Fluid is contained in a square cavity with three rigid walls (bottom, left and right) and one moving wall (top).

$$\frac{\partial U}{\partial X} + \frac{\partial V}{\partial Y} = 0$$

$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -\frac{\partial P}{\partial X} + \frac{1}{Re}\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right)$$

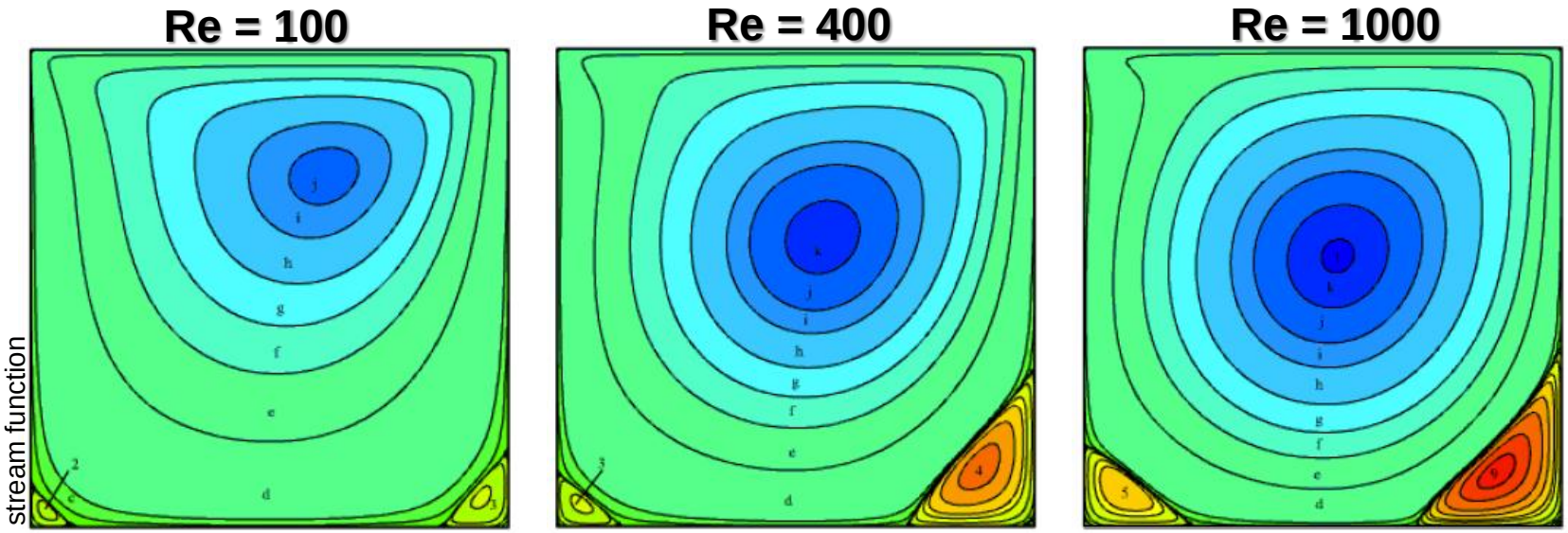
$$\frac{\partial V}{\partial \tau} + U\frac{\partial V}{\partial X} + V\frac{\partial V}{\partial Y} = -\frac{\partial P}{\partial Y} + \frac{1}{Re}\left(\frac{\partial^2 V}{\partial X^2} + \frac{\partial^2 V}{\partial Y^2}\right)$$

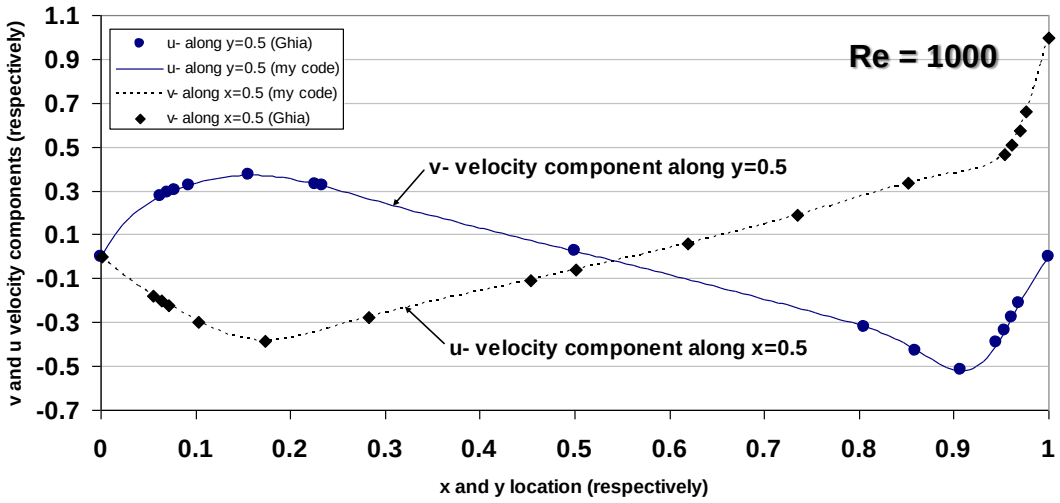


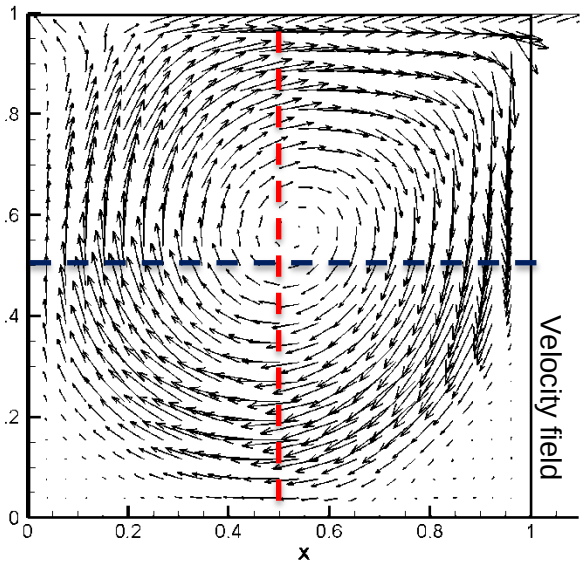
- Reynolds number is a non dimensional parameter that measure of the ratio of inertial forces to viscous forces:

$$Re = \frac{inertial\ force}{viscous\ force} = \frac{UL}{v}$$

Case 1: Lid Driven Cavity





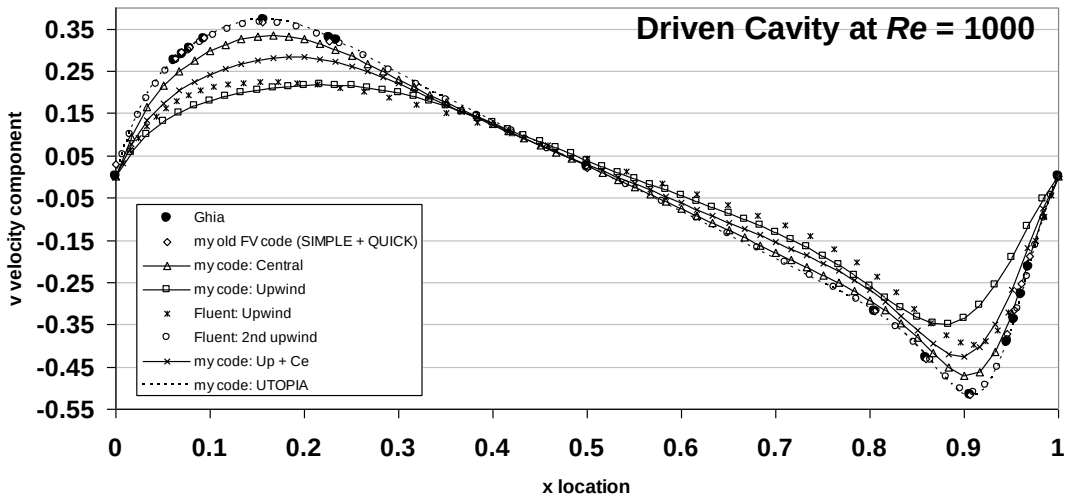


Accuracy improvements

$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -\frac{\partial P}{\partial X} + \frac{1}{Re}\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right)$$

- Upwind (central difference replaced by one sided-differences),
- Upwind + Central Difference,
- QUICK,
- UTOPIA:

```
float unew = uo[i][j];
if ((i==1) || (i>=(NX-1)))
{
  o1 = (
        (uo[i+1][j]-uo[i-1][j])-
        1.0f*sgn(unew)*(uo[i+1][j])
        -2.0f*uo[i][j]+uo[i-1][j]) /
  (2.0f*deltaX);
}
else
{
  if (unew>0.0f) dif=1.0f; else
    if (unew<0.0f) dif=-1.0f; else dif=0.0f;
    o1= ( (dif-1)/12 * uo[i+2][j]
         -(dif-2)/ 3 * uo[i+1][j]
         +(dif  )/ 2 * uo[i  ][j]
         -(dif+2)/ 3 * uo[i-1][j]
         +(dif+1)/12 * uo[i-2][j] ) / deltaX;
    }
float FUX = unew*o1;
```



Case 2: Natural Convection

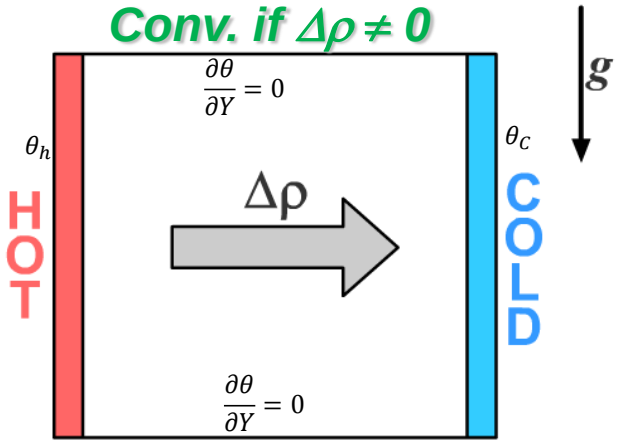
- Governing equations:

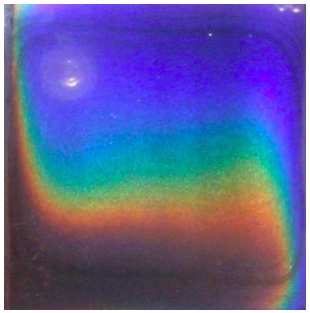
$$\frac{\partial U}{\partial X}+\frac{\partial V}{\partial Y}=0$$

$$\frac{\partial U}{\partial \tau}+U\frac{\partial U}{\partial X}+V\frac{\partial U}{\partial Y}=-\frac{\partial P}{\partial X}+Pr\left(\frac{\partial^2 U}{\partial X^2}+\frac{\partial^2 U}{\partial Y^2}\right)$$

$$\frac{\partial V}{\partial \tau}+U\frac{\partial V}{\partial X}+V\frac{\partial V}{\partial Y}=-\frac{\partial P}{\partial Y}+Pr\left(\frac{\partial^2 V}{\partial X^2}+\frac{\partial^2 V}{\partial Y^2}\right)+RaPr\theta$$

$$\frac{\partial \theta}{\partial \tau}+U\frac{\partial \theta}{\partial X}+V\frac{\partial \theta}{\partial Y}=\left(\frac{\partial^2 \theta}{\partial X^2}+\frac{\partial^2 \theta}{\partial Y^2}\right)$$




Temperature (TLC)

- Characteristic non dimensional variables:

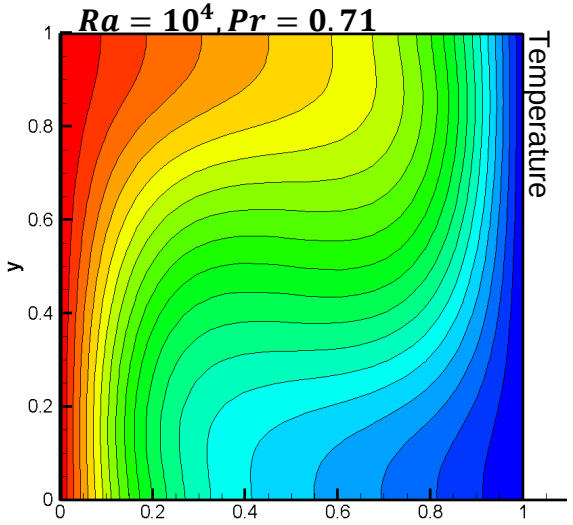
$$Pr=\frac{\nu}{\alpha}$$

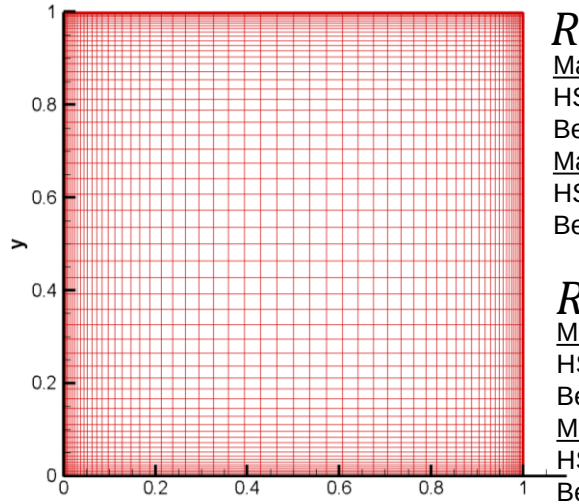
Prandtl number describes the relative strength of the diffusion of momentum to that of heat.

$$Ra=\frac{g\beta(\theta_h-\theta_c)l^3}{\alpha\nu}$$

Rayleigh number expresses the ratio of the buoyancy forces to viscous forces.

Case 2: Natural Convection





$Ra = 10^4$

Max of V- at Y = 0.5:

HSMAC: 19.56 at x = 0.1114 err 0.2906%

Benchmark: 19.617 at x = 0.119

Max of U- at X = 0.5:

HSMAC: 16.108 at y = 0.8324 err 0.4327%

Benchmark: 16.178 at y = 0.823

$Ra = 10^6$

Max of V- at Y = 0.5:

HSMAC: 219.5 at x = 0.03622 err 0.0638%

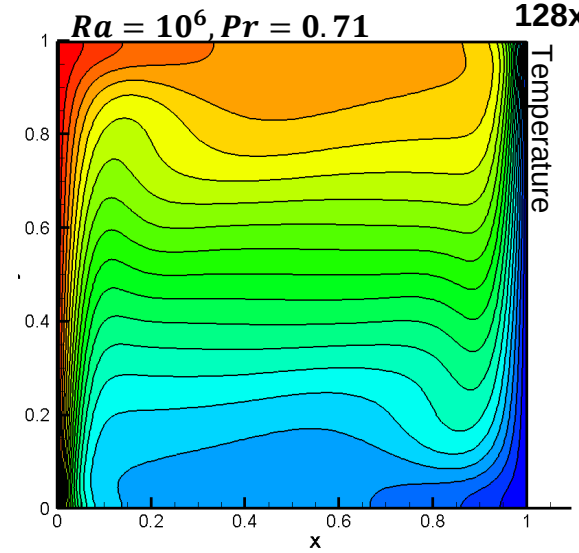
Benchmark: 219.36 at x = 0.03790

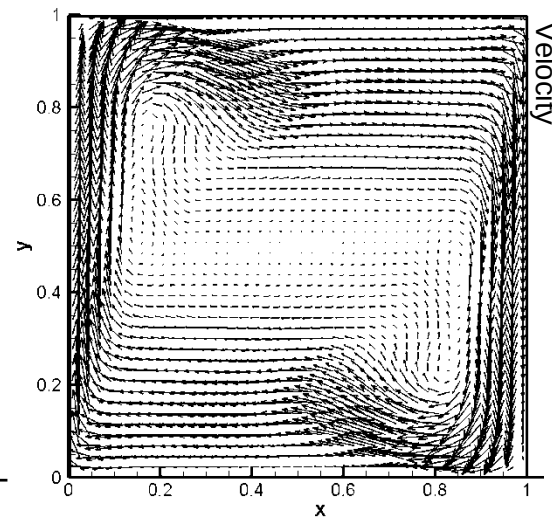
Max of U- at X = 0.5:

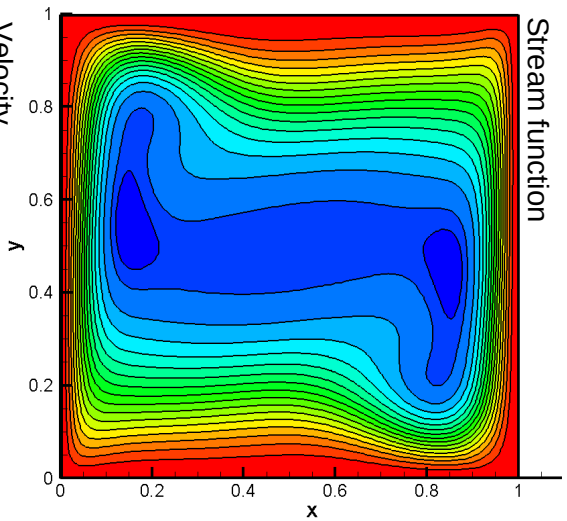
HSMAC: 64.492 at y = 0.85 err 0.2135%

Benchmark: 64.630 at y = 0.850

128x128







- Variables to be compared: temperature and velocity profiles, stream function, Nusselt number

Case 3: Magneto-thermal convection

Magnetic body force acting on electrically non-conducting materials:

$$\vec{f}_{magnetic} = \frac{\chi\rho}{2\mu_m}\vec{\nabla}b^2$$

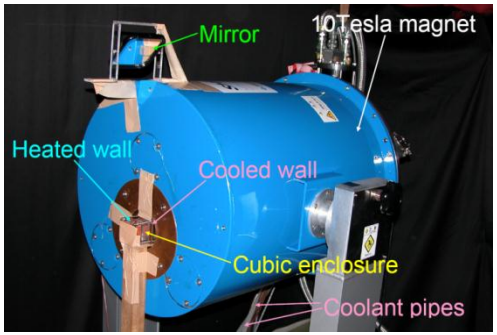
Magnetic suceptibility of paramagnetic substances (χ) is inversly proportional to its absolute temperature θ – Curie law:

$$\chi = \frac{const}{\theta}$$

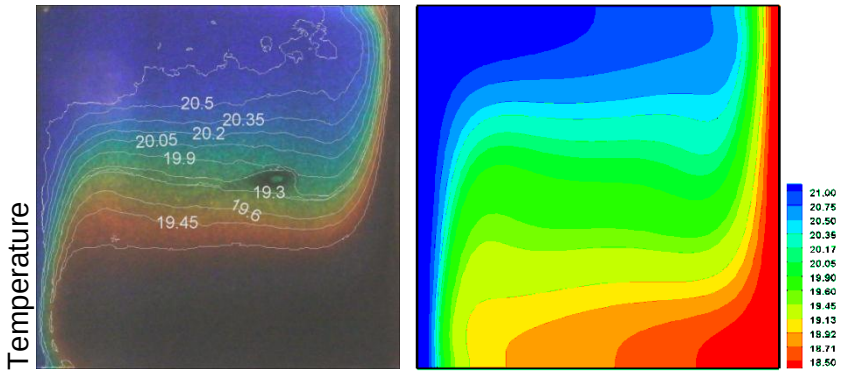
Using Curie's law we can obtain the magnetic force (similar to Bousinessq approximation):

$$\vec{f}_{mb} = -\left(1 + \frac{1}{\theta_0\beta}\right)\frac{\chi_0\rho_0\beta(\theta-\theta_0)}{2\mu_m}\vec{\nabla}b^2$$

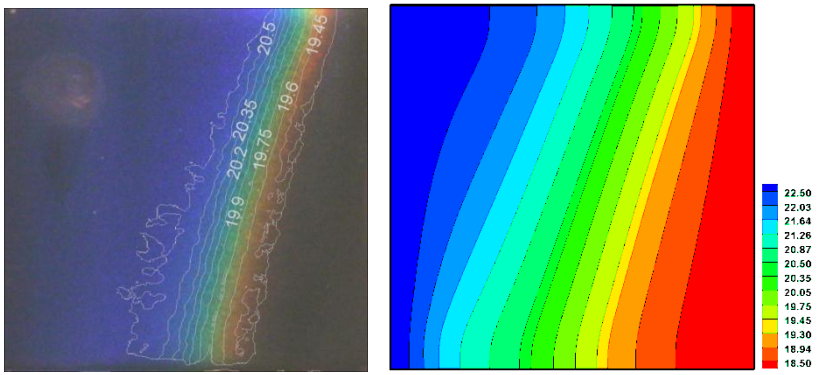
$\theta < \theta_0$: ***attractive force***
$\theta > \theta_0$: ***repulsive force***



- Suppression of the convection:



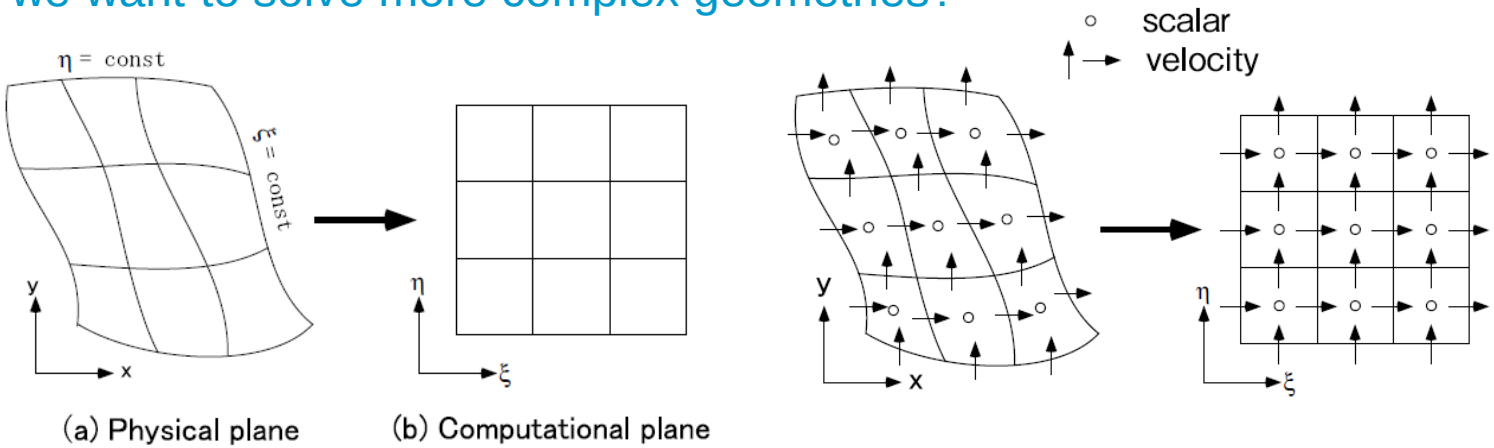
(a) 0 [T], $\Delta\theta$ = 3.44 [K]



(b) 10 [T], $\Delta\theta$ = 5.27 [K]

Boundary Fitted Coordinates (BFC)

- If we want to solve more complex geometries?



(a) Physical plane (b) Computational plane

- We need to generate a smooth orthogonal grid system :
  - Specify boundaries in the physical plane,
  - Solve:

$$\begin{cases} \xi &= \xi(x,y) \\ \eta &= \eta(x,y) \end{cases} \qquad \begin{cases} \xi_{xx} + \xi_{yy} &= P'(\xi,\eta) \\ \eta_{xx} + \eta_{yy} &= Q'(\xi,\eta) \end{cases}$$

  - Or easier:

$$\begin{cases} x &= x(\xi,\eta) \\ y &= y(\xi,\eta) \end{cases}$$

$$\begin{cases} \alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} &= -J^2(P'(\xi,\eta)x_\xi + Q'(\xi,\eta)x_\eta) = P(\xi,\eta) \\ \alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} &= -J^2(P'(\xi,\eta)y_\xi + Q'(\xi,\eta)y_\eta) = Q(\xi,\eta) \end{cases}$$

$$x_\xi = \frac{x_{i+1,j} - x_{i-1,j}}{2\Delta\xi}$$

$$x_\eta = \frac{x_{i,j+1} - x_{i,j-1}}{2\Delta\eta}$$

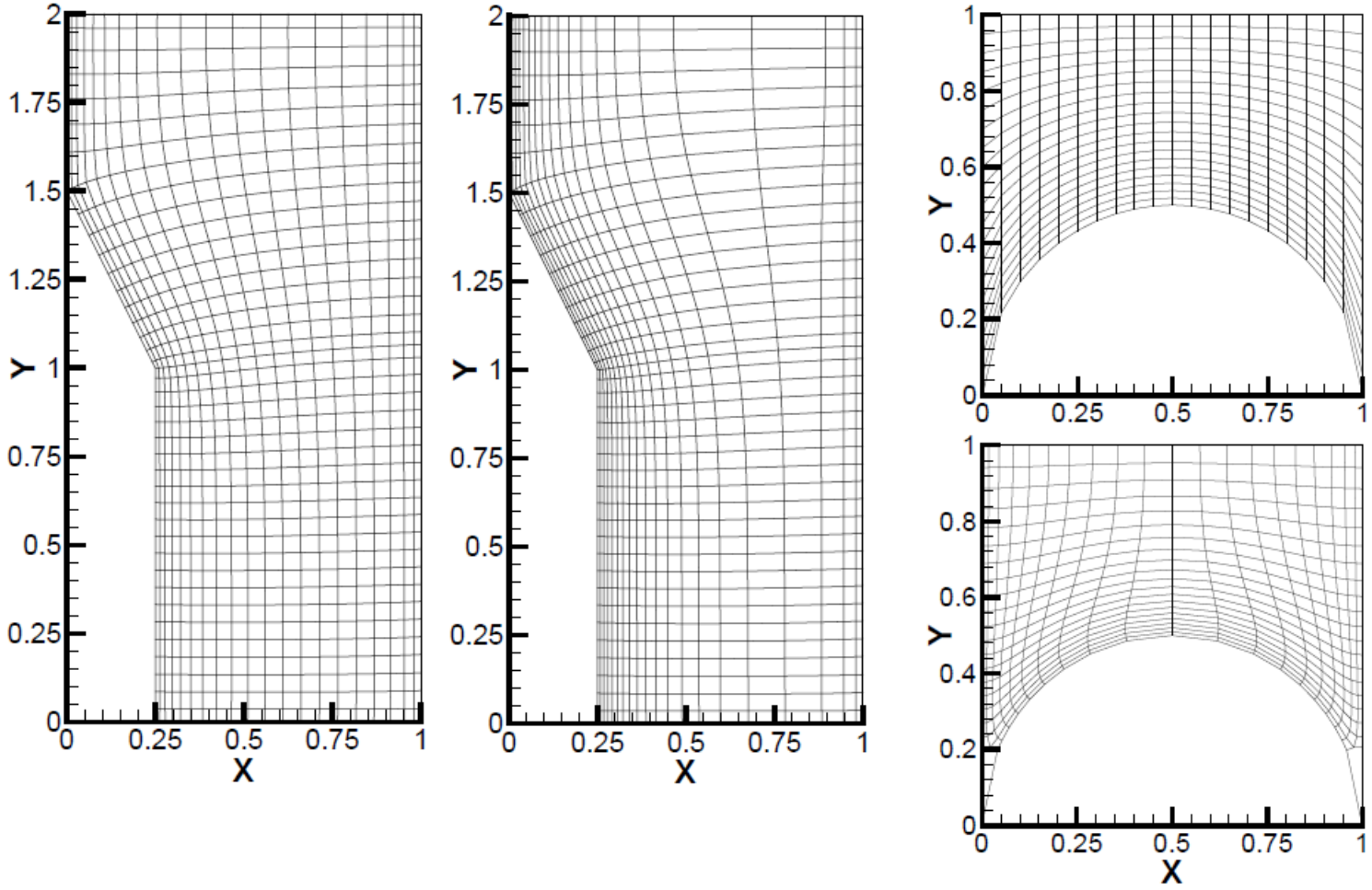
$$x_{\xi\xi} = \frac{x_{i-1,j} - 2x_{i,j} + x_{i+1,j}}{\Delta\xi^2}$$

$$x_{\eta\eta} = \frac{x_{i,j-1} - 2x_{i,j} + x_{i,j+1}}{\Delta\eta^2}$$

$$\begin{cases} \alpha &= {x_\eta}^2 + {y_\eta}^2 \\ \beta &= x_\xi x_\eta + y_\xi y_\eta \\ \gamma &= {x_\xi}^2 + {y_\xi}^2 \end{cases}$$

Boundary Fitted Coordinates (BFC)

- Sample grids



Boundary Fitted Coordinates (BFC)

- Now, if we have our continuity equation in physical plane:

$$\frac{\partial U}{\partial X}+\frac{\partial V}{\partial Y}=0$$

with transformation between physical and computational plane

$$\begin{cases} \xi &= \xi(x,y) \\ \eta &= \eta(x,y) \end{cases}$$

- Using chain rules of differential calculus, we have:

$$\frac{\partial}{\partial X}=\left(\frac{\partial}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial X}\right)+\left(\frac{\partial}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial X}\right)$$

$$\frac{\partial}{\partial Y}=\left(\frac{\partial}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial Y}\right)+\left(\frac{\partial}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial Y}\right)$$

$$\begin{bmatrix} \frac{\partial}{\partial X} \\ \frac{\partial}{\partial Y} \end{bmatrix}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix}$$

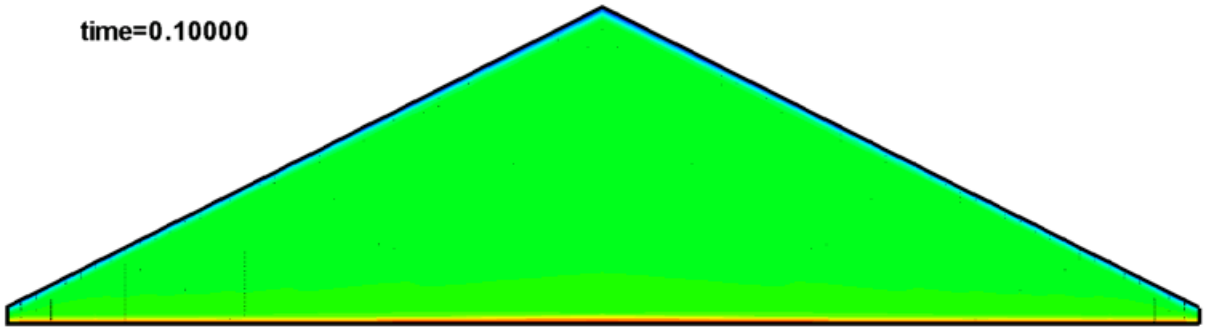
$$\frac{\partial f}{\partial X}=a_{11}\frac{\partial f}{\partial \xi}+a_{12}\frac{\partial f}{\partial \eta}$$

$$\frac{\partial f}{\partial Y}=a_{21}\frac{\partial f}{\partial \xi}+a_{22}\frac{\partial f}{\partial \eta}$$

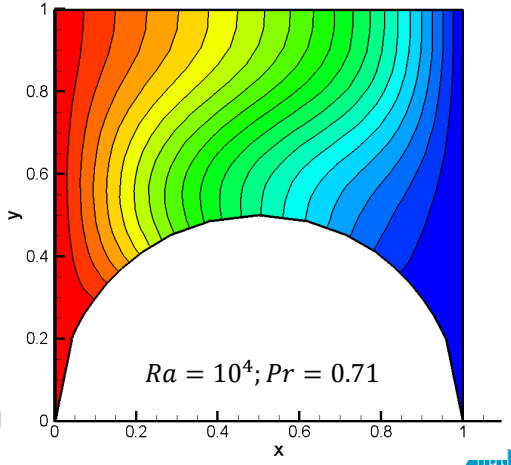
$$J\equiv\frac{\partial(X,Y)}{\partial(\xi,\eta)}\equiv\begin{vmatrix} X_\xi & X_\eta \\ Y_\xi & Y_\eta \end{vmatrix}=X_\xi Y_\eta - X_\eta Y_\xi$$

the Jacobian of the transformation represents the ratio of the volume of a grid call in physical space to one in computational space.

$$\frac{\partial^2 f}{\partial X^2}=\frac{\partial}{\partial X}\left(\frac{\partial}{\partial X}\right)=a_{11}\frac{\partial}{\partial \xi}\left(a_{11}\frac{\partial f}{\partial \xi}+a_{12}\frac{\partial f}{\partial \eta}\right)+a_{12}\frac{\partial}{\partial \eta}\left(a_{11}\frac{\partial f}{\partial \xi}+a_{12}\frac{\partial f}{\partial \eta}\right)$$

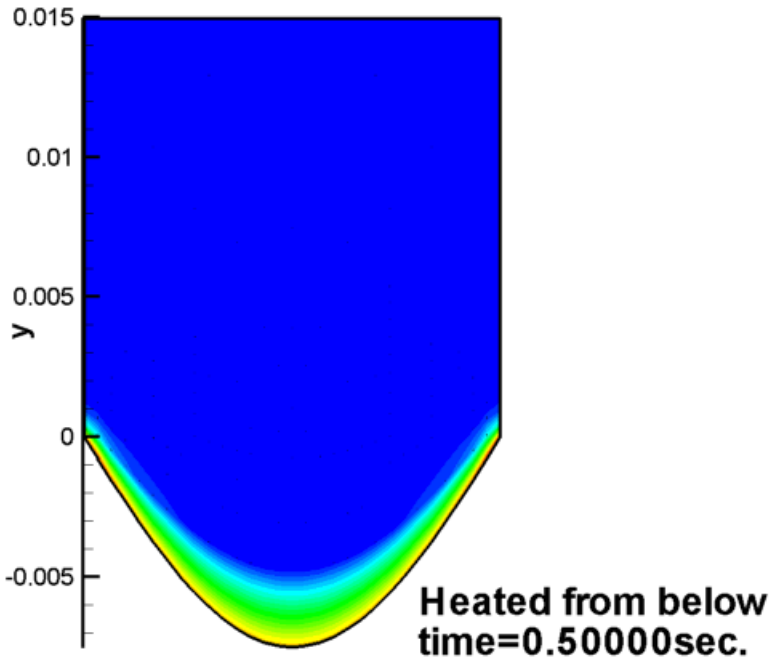


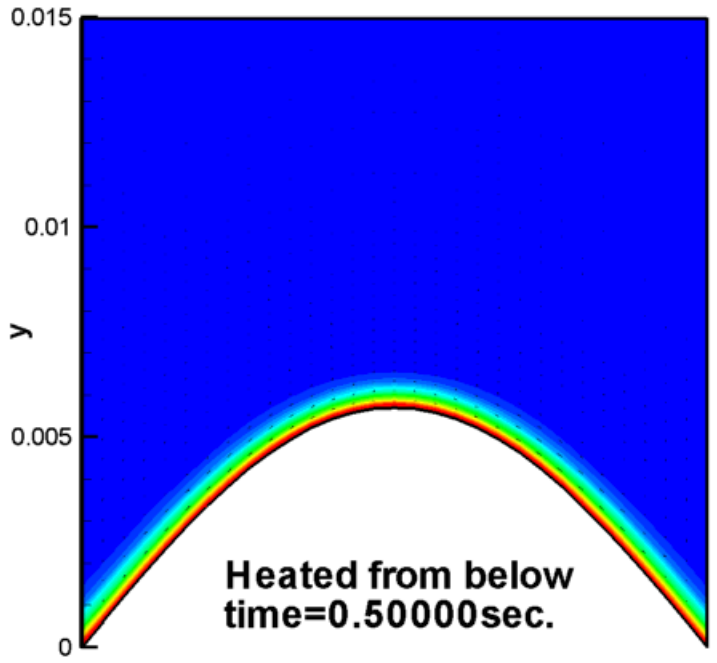
attic space simulation



Boundary Fitted Coordinates (BFC)

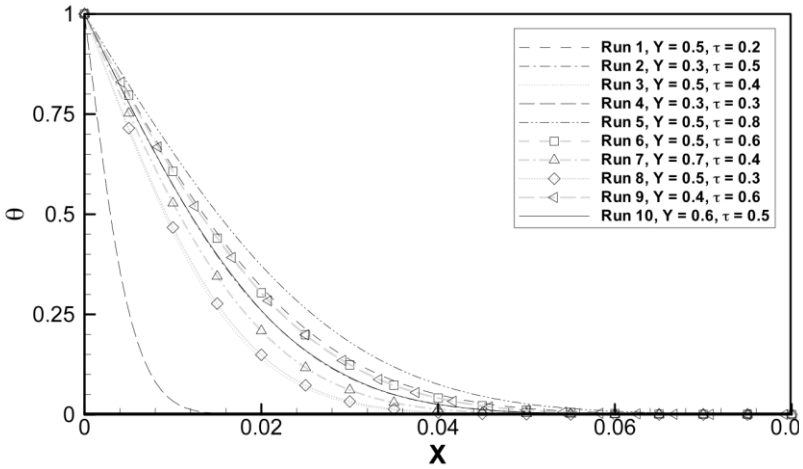
- Two more sample simulations:

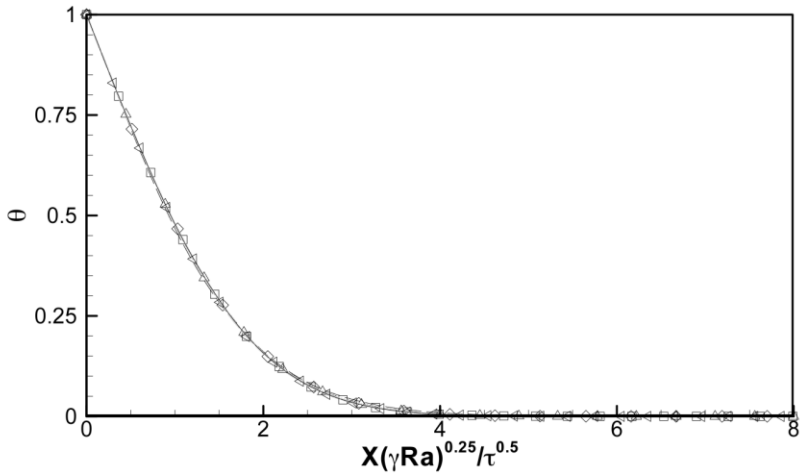




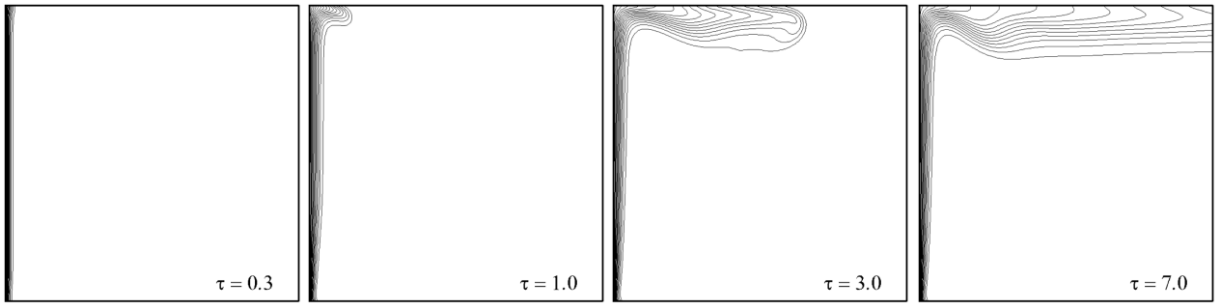
Applications: scaling analysis

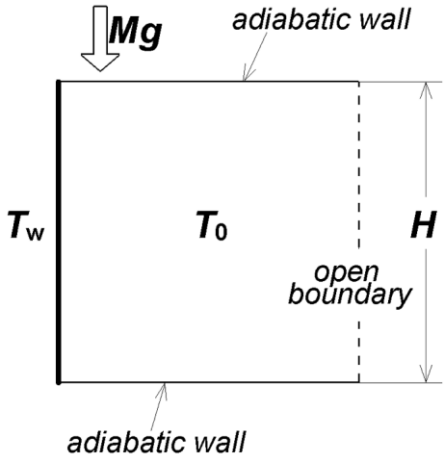
- Used to predict behaviour of the fluid flow for different cases.
- Approach: by comparing terms of governing equations.





- Transient flow:



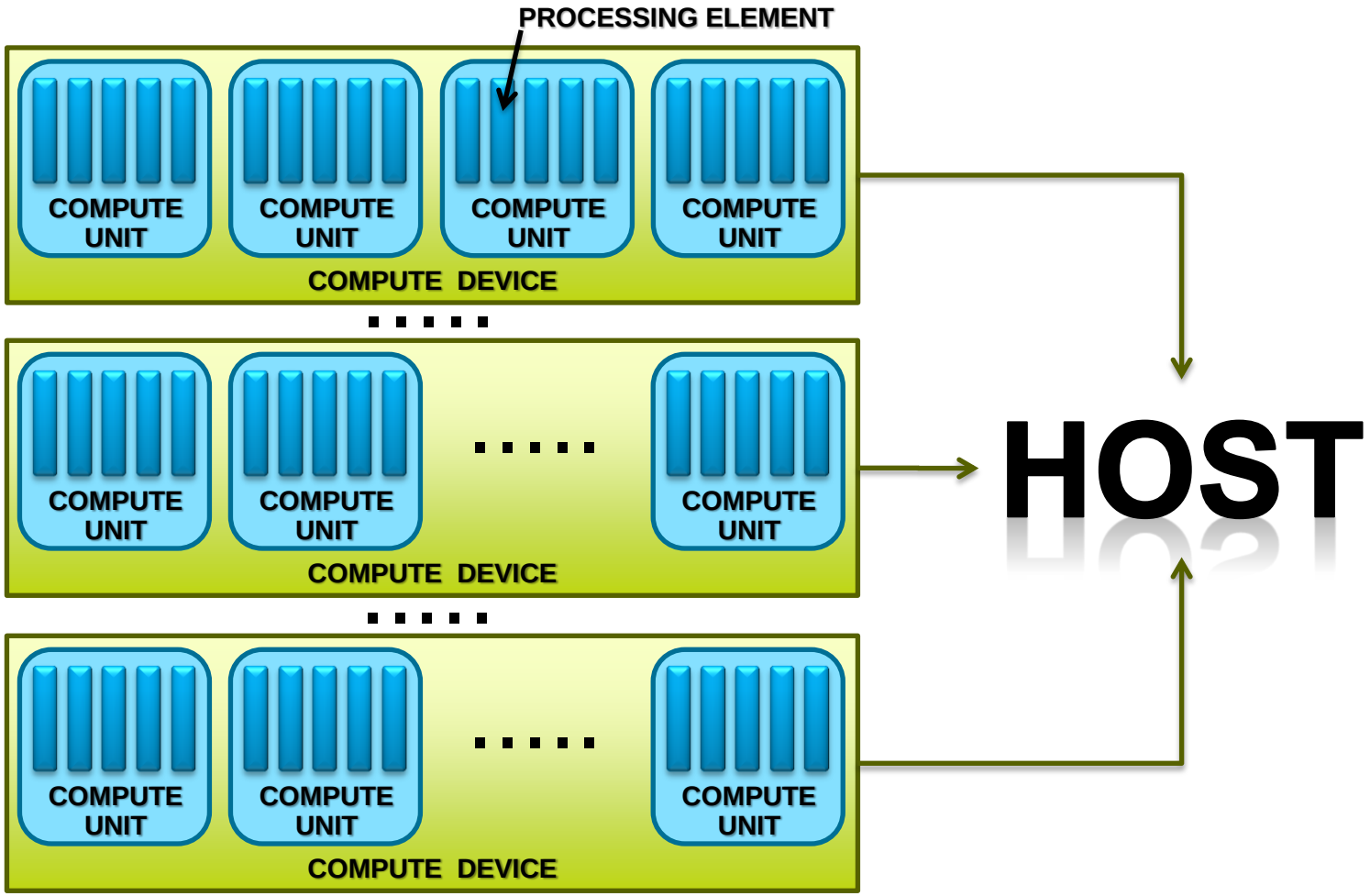


HSMAC on GPU

OpenCL™

- **OpenCL** (Open Compute Language): an open, royalty-free standard for parallel programming of heterogeneous systems that include multi-core processors (CPUs), graphics processing units (GPUs), and other accelerators such as Cell and digital signal processors (DSPs).
- **OpenCL** has complex platform and device management model that reflects its support for multi-platform and multi-vendor portability.
- Uniform programming environment to write efficient, portable code for HPC servers, desktop computer systems and handheld devices.
- The standard includes an **API** for coordinating execution between devices and a cross-platform parallel programming language.
- Initiated by **Apple** (now specification editor) and developed by the **Khronos Group**.

OpenCL Platform Model



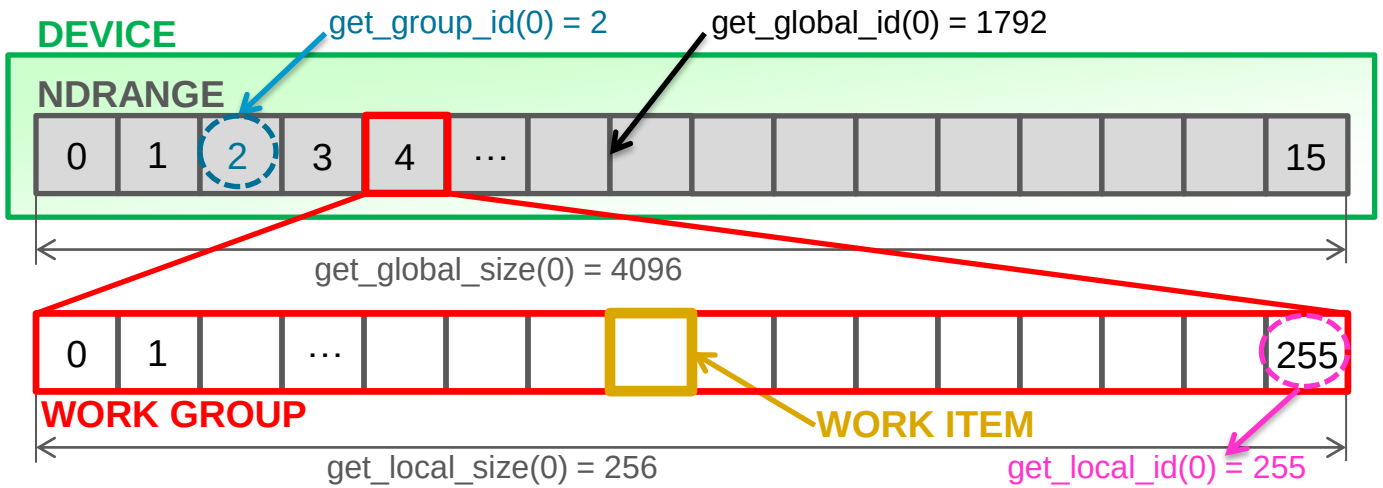
Tested GPUs

- GPU Features…

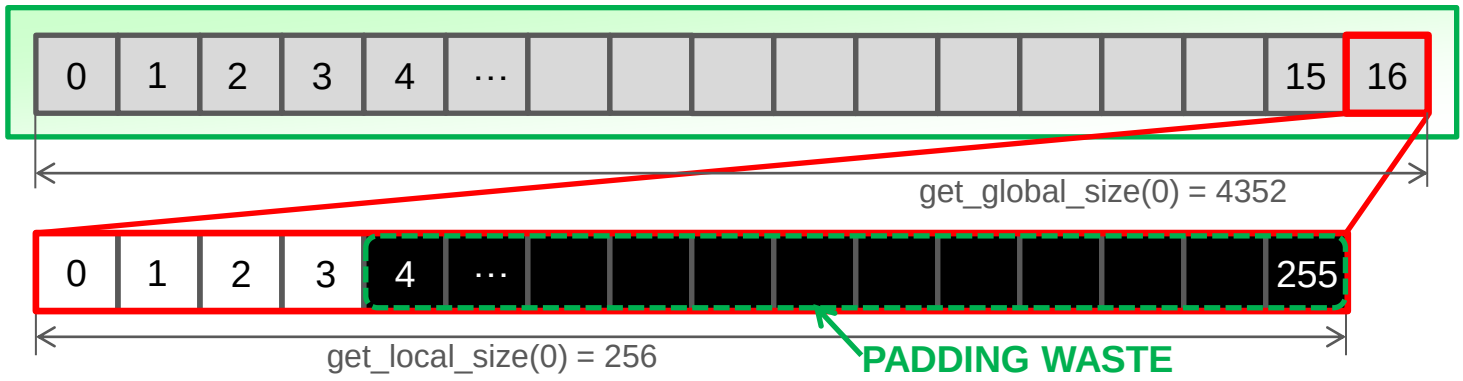
	GT 330M	QFX4600	QFX4800	460GTX	S2050
Streaming Multiprocessors (SM)	6	12	24	7	14
Streaming Processors (SP) / Cores	48	96	192	224	448
Registers per SM	16K	8K	16K	32KB	32KB
Shared Memory per SM	16KB	16KB	16KB	48KB	48KB
Processor Clock [MHz]	990	1200	1204	810	1147
Work item size	512/512/64	512/512/64	512/512/64	1024/1024/64	1024/1024/64
Work group size	512	512	512	1024	1024
CUDA Compute Capability	1.2	1.0	1.3	2.x	2.x

OpenCL execution model 1-D

- Number of work items = 4096: (A kernel is executed in each point of a problem domain)



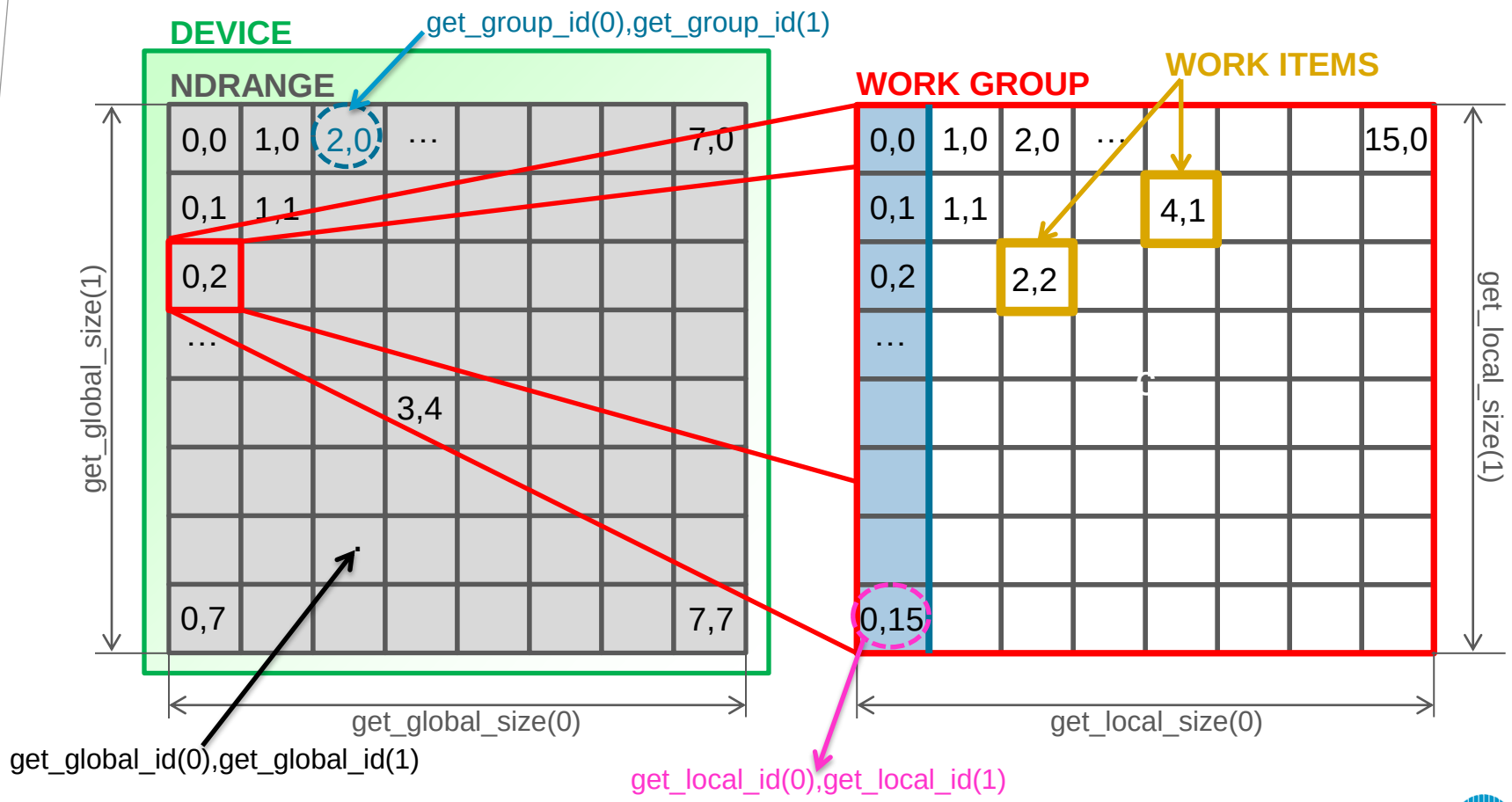
- Number of work items = 4100:



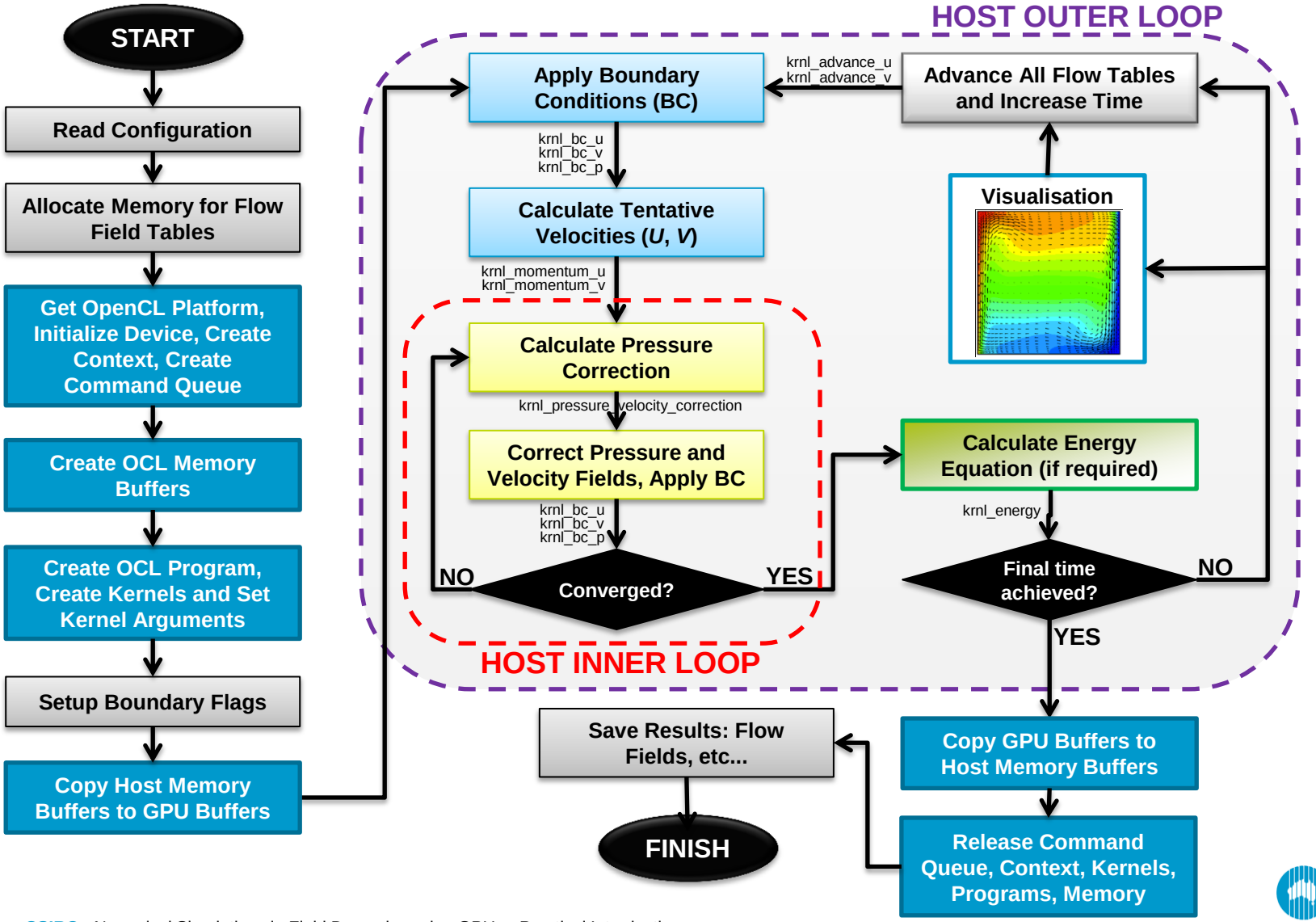
OpenCL execution model 2-D

- Number of work items to execute 128 x 128 = 16384:

(A kernel is executed in each point of a problem domain)



Flow Chart of the OpenCL Fluid Solver



Before we start…

- Get OpenCL platform and initialise computing devices:

```
// GET OPENCL PLATFORM

std::vector<cl_platform_id> platformIDs;
cl_uint nPlatforms;

/// get number of OpenCL platforms available
cl_int err = clGetPlatformIDs(0, NULL, &nPlatforms);
if (err != CL_SUCCESS)
{
  // ...
}
platformIDs.resize(nPlatforms);

// get all OpenCL platform IDs
err = clGetPlatformIDs(nPlatforms, &platformIDs[0], NULL);
if (err != CL_SUCCESS)
{
  // ...
}
```

```
// GET OPENCL DEVICES

std::vector<cl_device_id> deviceIDs;
cl_uint nDevices;

err = clGetDeviceIDs(platformIDs[selectedPlatform],
                     CL_DEVICE_TYPE_GPU, 0, NULL, &nDevices);
if (err != CL_SUCCESS)
{
  // ...
}
deviceIDs.resize(nDevices);

// get device IDs of selected platform
err = clGetDeviceIDs(platformIDs[selectedPlatform],
                     CL_DEVICE_TYPE_GPU, nDevices, &deviceIDs[0], NULL);
if (err != CL_SUCCESS)
{
  // ...
}
```

- Create context and create command queue.

```
// CREATE OPENCL CONTEXT

cl_int err;
cl_context context =
  clCreateContext(0, deviceIDs.size(),
                  &deviceIDs[0], NULL, NULL, &err);
if (err != CL_SUCCESS)
{
  // ...
}
```

```
// CREATE COMMAND QUEUE

std::vector<cl_command_queue> commandQueues;
commandQueues.resize(deviceIDs.size());

for (int i=0; i<deviceIDs.size(); i++)
{
  commandQueues[i] =
    clCreateCommandQueue(
      context, deviceIDs[i], CL_QUEUE_PROFILING_ENABLE, &err);
  if (err != CL_SUCCESS)
  {
    // ...
  }
}
```

Before we start…

- Create OpenCL memory buffers:

```
// CREATE MEMORY BUFFER

// flags: CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY
//        CL_MEM_READ_WRITE, ...
// size:  NX2*NY2

cl_mem buffer;

//
// create the input/output arrays in device memory
//
buffer = clCreateBuffer(context, flags, size, NULL, &err);
if (buffer == 0 || err != CL_SUCCESS)
{
  // ...
}
```

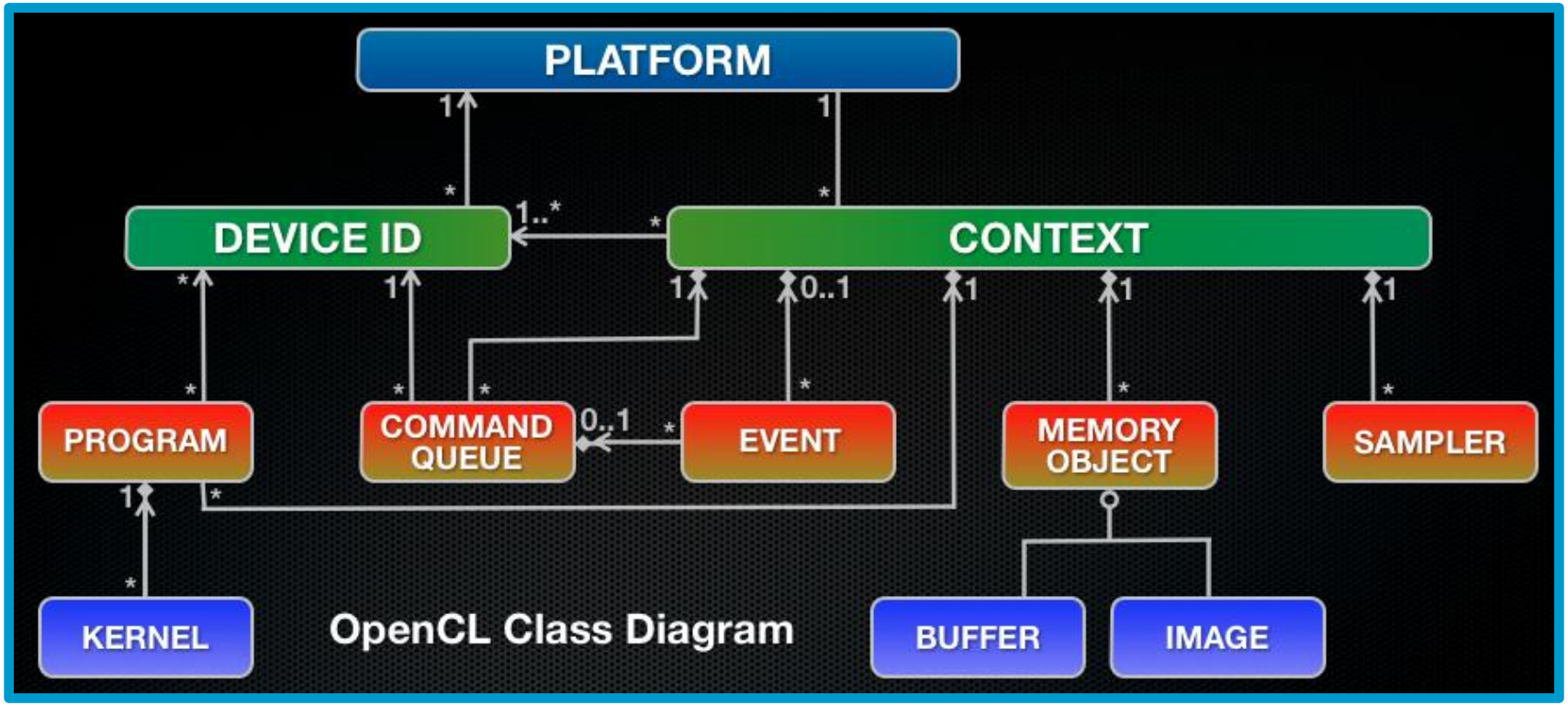
- Create OpenCL program, kernel and set kernel arguments:

```
// CREATE PROGRAM WITH SOURCE

cl_int err;
cl_program clProgram;

// create the compute program from the source buffer
clProgram = clCreateProgramWithSource(context, 1,
                (const char **)&programString, NULL, &err);
if (!clProgram)
{
  // ...
}

// build the program executable
//
err = clBuildProgram(clProgram, 0, NULL, NULL, NULL, NULL);
if (err != CL_SUCCESS)
{
  size_t len;
  char buffer[2048];
  clGetProgramBuildInfo(clProgram, deviceIDs[0],
                        CL_PROGRAM_BUILD_LOG, sizeof(buffer),
                        buffer, &len);
  // ...
}
```

```
// CREATE OPENCL KERNEL

cl_kernel kernel;

// create the compute kernel in the program we wish to run
kernel = clCreateKernel(clProgram, kernelName, &err);
if (!kernel || err != CL_SUCCESS)
{
  // ...
}

// SET ARGUMENTS FOR COMPUTE KERNEL

err  = clSetKernelArg(kernel, index, size, value);

if (err != CL_SUCCESS)
{
  // ...
}
```

OpenCL Class Diagram

- UML class diagram of OpenCL: relationships between classes,
- Solid diamonds (aggregations), inheritance (open arrowhead), cardinality: ***** many, 1 one and only one, 0..1 optionally one, 1..* one or more.



Reconstructed from Figure 2.1 of the Khronos OpenCL Specification.

ComputeEngine Class

- Example:

```
ComputeEngine compute;

// initialize OpenCL
compute.init(CL_DEVICE_TYPE_GPU);

// create device memory objects
// u
compute.createBuffer("un", CL_MEM_READ_WRITE, sizeof(float)*NX2*NY2);
compute.createBuffer("uo", CL_MEM_READ_WRITE, sizeof(float)*NX2*NY2);
compute.createBuffer("ug", CL_MEM_READ_WRITE, sizeof(int)*NX2*NY2);
// v
compute.createBuffer("vn", CL_MEM_READ_WRITE, sizeof(float)*NX2*NY2);
compute.createBuffer("vo", CL_MEM_READ_WRITE, sizeof(float)*NX2*NY2);
compute.createBuffer("vg", CL_MEM_READ_WRITE, sizeof(int)*NX2*NY2);
// p
compute.createBuffer("p", CL_MEM_READ_WRITE, sizeof(float)*NX2*NY2);
compute.createBuffer("pg", CL_MEM_READ_WRITE, sizeof(int)*NX2*NY2);

// create OpenCL program
compute.createProgramFromFile("hsmac", "hsmac2.cl");

// kernel: advance of u (2-D)
compute.createKernel("hsmac", "ocl_hsmac_advance_u2");
compute.setKernelArgument("ocl_hsmac_advance_u2", 0, sizeof(cl_mem), &compute.memoryObjects["un"]);
compute.setKernelArgument("ocl_hsmac_advance_u2", 1, sizeof(cl_mem), &compute.memoryObjects["uo"]);
compute.setKernelArgument("ocl_hsmac_advance_u2", 2, sizeof(unsigned int), &NX2);
compute.setKernelArgument("ocl_hsmac_advance_u2", 3, sizeof(unsigned int), &NY2);

// ...
// ...
// ...

compute.writeBuffer(0, "ug", 0, sizeof(int)*NX2*NY2, &ug_ocl[0]);
compute.writeBuffer(0, "vg", 0, sizeof(int)*NX2*NY2, &vg_ocl[0]);
compute.writeBuffer(0, "pg", 0, sizeof(int)*NX2*NY2, &pg_ocl[0]);
```

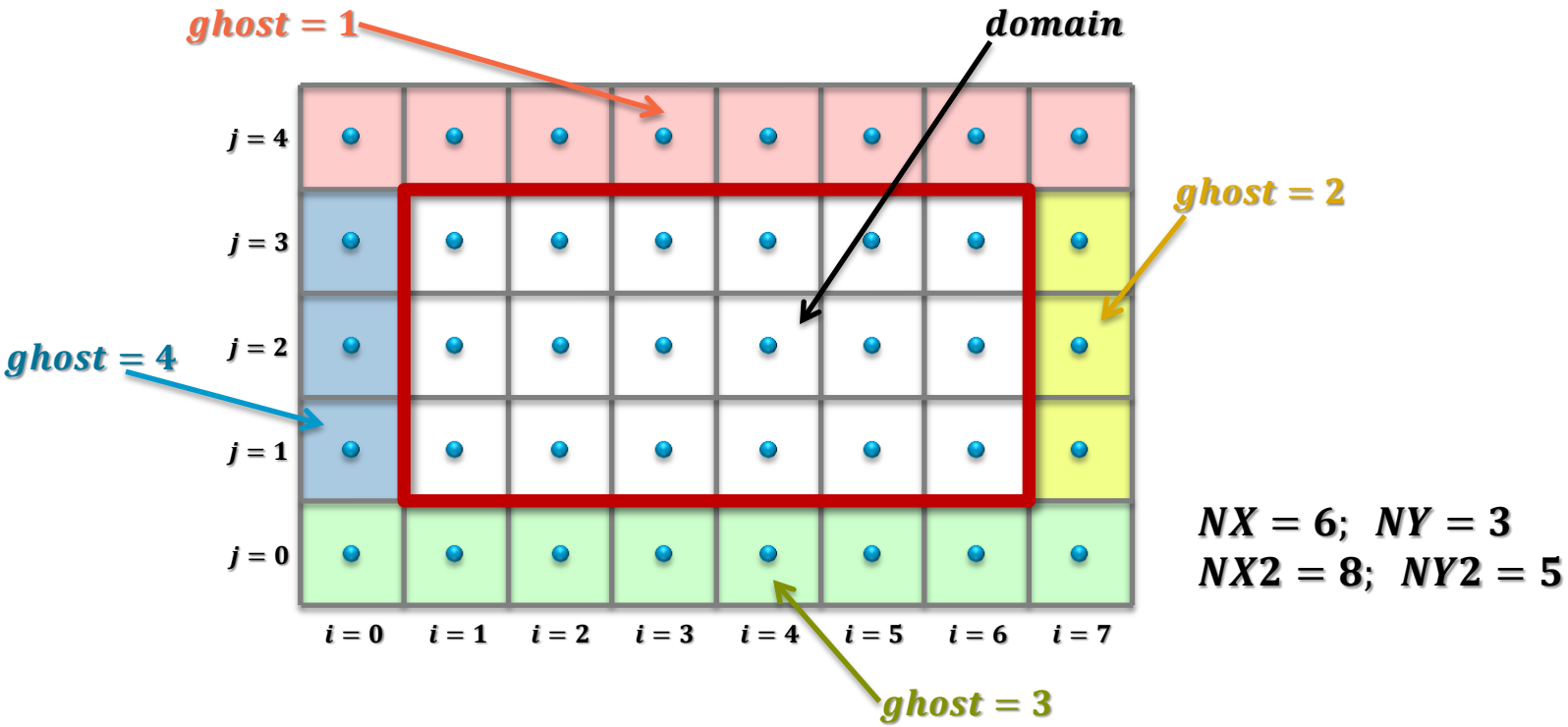
```
__kernel void ocl_hsmac_advance_u2(
  __global float* un,
  __global float* uo,
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int i = get_global_id(0);
  unsigned int j = get_global_id(1);

  if (i<NX2 && j<NY2)
  {
    uo[i+j*NX2] = un[i+j*NX2];
  }
}
```

- Kernels call:

```
// advance flow fields
compute.runKernel("ocl_hsmac_advance_u2", 0, 2, workSizeGlobal, workSizeLocal);
compute.runKernel("ocl_hsmac_advance_v2", 0, 2, workSizeGlobal, workSizeLocal);
clFinish(compute.commandQueues[0]);
```

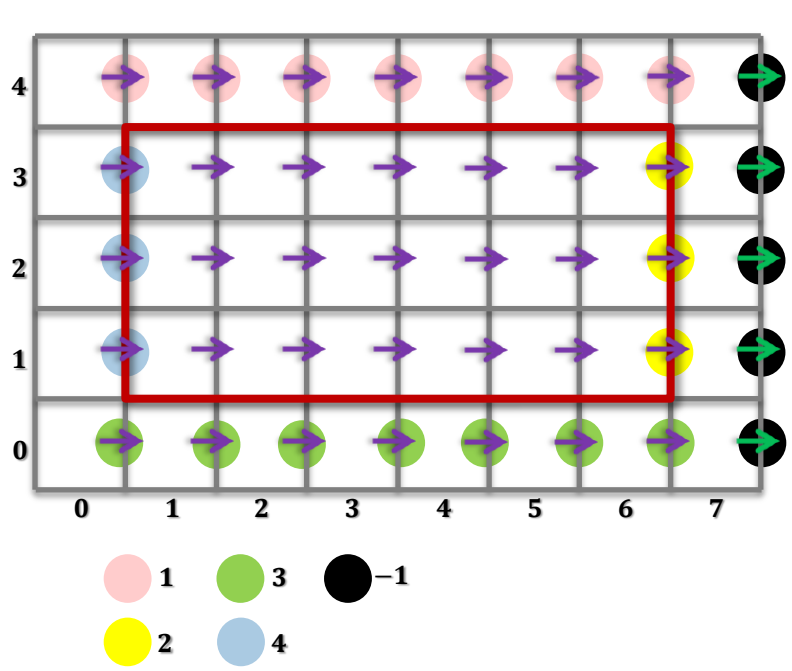
Boundary Conditions on GPU

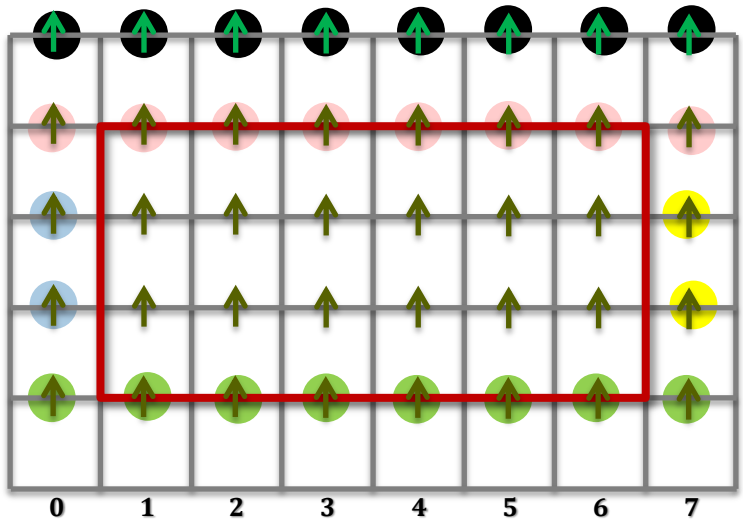
- Setup boundary conditions
  - Create “ghost” cells for velocity, pressure, temperature and concentration fields.
  - Below: ghost cells to assist scalar computations.



Boundary conditions on GPU

- Velocity boundary conditions
  - For simplicity, the memory buffer for vectors is of the same size as for scalar variables.
  - For horizontal velocity component there is one column of waste.
  - For vertical velocity components there is one row of waste.

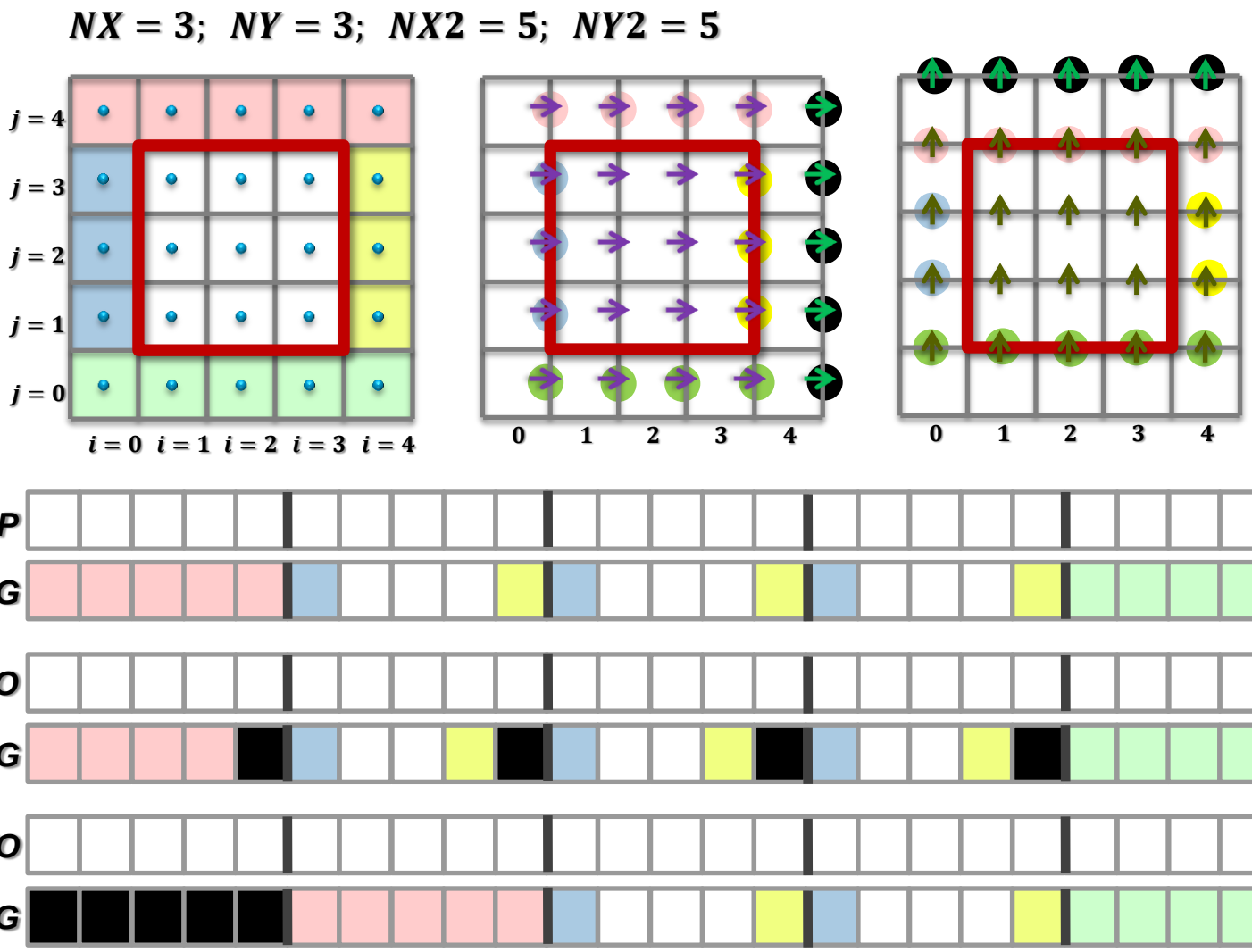




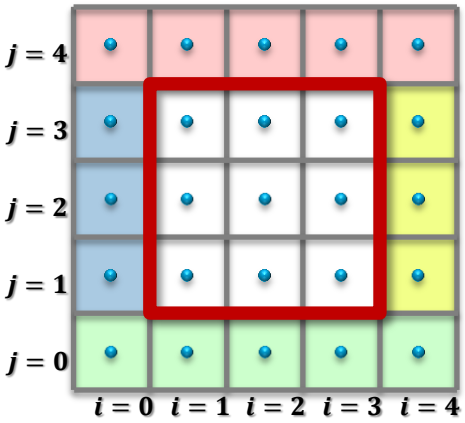
$NX = 6;\ NY = 3$

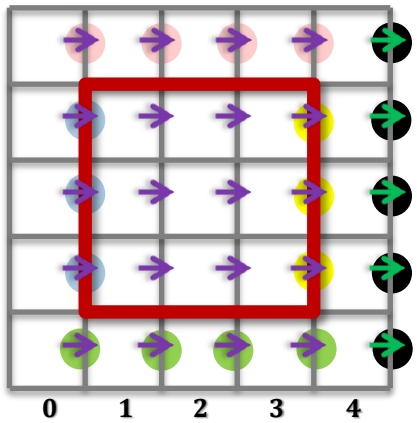
$NX2 = 8;\ NY2 = 5$

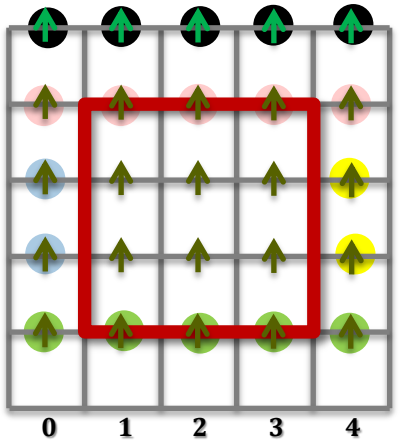
Memory Buffers Structure



Apply boundary conditions








```
__kernel void ocl_hsmac_bc_p(
  __global float* p,
  __global int* pg,
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int x = get_global_id(0);
  unsigned int y = get_global_id(1);

  unsigned int i = x+y*NX2;

  int ghostp = pg[i];

  if (ghostp == 4) // LEFT BOUNDARY
  {
    p[i] = p[i+1];
  } else
  if (ghostp == 2) // RIGHT BOUNDARY
  {
    p[i] = p[i-1];
  } else
  if (ghostp == 1) // TOP BOUNDARY
  {
    p[i] = p[i-NX2];
  } else
  if (ghostp == 3) // BOTTOM BOUNDARY
  {
    p[i] = p[i+NX2];
  }
}
```

```
__kernel void ocl_hsmac_bc_u(
  __global float* un,
  __global int* ug,
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int x = get_global_id(0);
  unsigned int y = get_global_id(1);

  unsigned int i = x+y*NX2;

  int ghostu = ug[i];

  if (ghostu == 4) // LEFT BOUNDARY
  {
    un[i] = 0.0f;
  } else
  if (ghostu == 2) // RIGHT BOUNDARY
  {
    un[i] = 0.0f;
  } else
  if (ghostu == 1) // TOP BOUNDARY
  {
    un[i] = 2.0f-un[i-NX2];
  } else
  if (ghostu == 3) // BOTTOM BOUNDARY
  {
    un[i] = -un[i+NX2];
  }
}
```

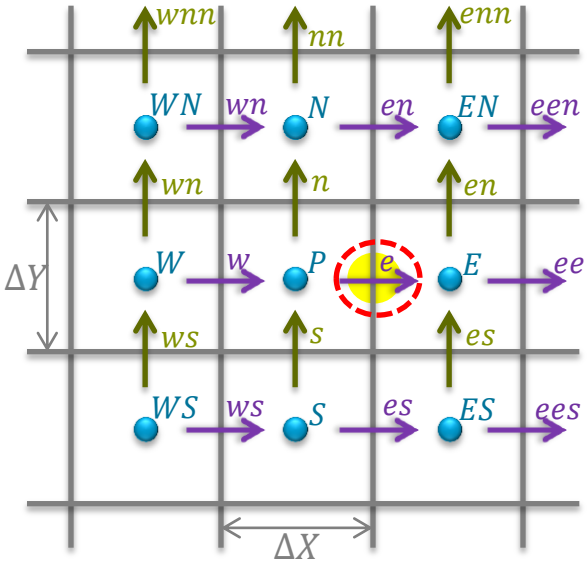
```
__kernel void ocl_hsmac_bc_v(
  __global float* vn,
  __global int* vg,
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int x = get_global_id(0);
  unsigned int y = get_global_id(1);

  unsigned int i = x+y*NX2;

  int ghostv = vg[i];

  if (ghostv == 4) // LEFT BOUNDARY
  {
    vn[i] = -vn[i+1];
  } else
  if (ghostv == 2) // RIGHT BOUNDARY
  {
    vn[i] = -vn[i-1];
  } else
  if (ghostv == 1) // TOP BOUNDARY
  {
    vn[i] = 0;
  } else
  if (ghostv == 3) // BOTTOM BOUNDARY
  {
    vn[i] = 0;
  }
}
```

Calculate tentative velocity component U

- From *X*- momentum equation

$$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -A\frac{\partial P}{\partial X} + B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right)$$

$$U_{i,j}^{k+1} = U_{i,j}^{k} + \Delta\tau(-FUX - FUY - PRESSU + DIFFU)$$



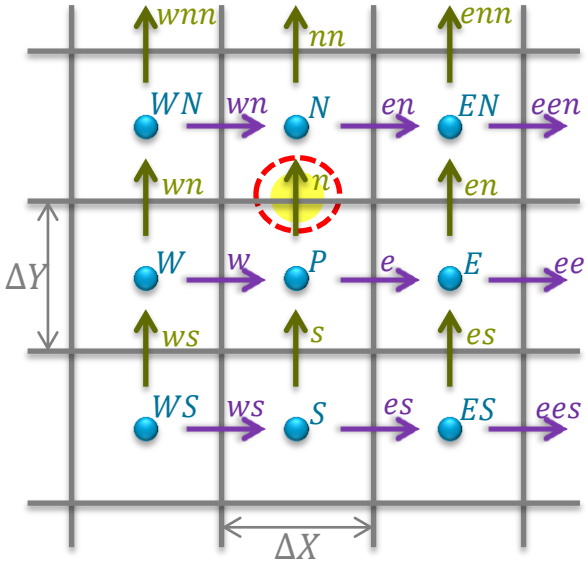

```
__kernel void ocl_hsmac_momentum_u(
  __global float* un, // velocity-x new
  __global float* uo, // velocity-x old
  __global float* vo, // velocity-y old
  __global float* p,  // pressure
  __global int* ug,   // boundaries
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int x = get_global_id(0);
  unsigned int y = get_global_id(1);
  unsigned int i = x+y*NX2;

  if (x<NX2 && y<NY2 && ug[i]==0)
  {
    float deltaTime = DELTA_TIME;

    float deltaX = DELTA_X;
    float deltaY = DELTA_Y;

    float udxdx = (((uo[i+1]-uo[i])/deltaX)-
                   ((uo[i]-uo[i-1])/deltaX)
                  )/deltaX;

    float udydy = (((uo[i+NX2]-uo[i])/deltaY)-
                   ((uo[i]-uo[i-NX2])/deltaY)
                  )/deltaY;

    float DIFFU = BB*(udxdx+udydy);

    float PRESSU = AA*(p[i+1]-p[i])/deltaX;

    float FUX = uo[i]*
          ((uo[i+1]-uo[i-1])/(2.0f*deltaX));

    float FUY =
    ((vo[i]+vo[i+1]+vo[i-NX2]+vo[i+1-NX2])/4.0f)*
    ((uo[i+NX2]-uo[i-NX2])/(2.0f*deltaY));

    un[i] = uo[i] +
            deltaTime*(-FUX-FUY-PRESSU+DIFFU);
  }
}
```

Calculate tentative velocity component V

- From *X*- momentum equation

$$\frac{\partial V}{\partial \tau} + U\frac{\partial V}{\partial X} + V\frac{\partial V}{\partial Y} = -A\frac{\partial P}{\partial Y} + B\left(\frac{\partial^2 V}{\partial X^2} + \frac{\partial^2 V}{\partial Y^2}\right)$$

$$V_{i,j}^{\,k+1} = V_{i,j}^{\,k} + \Delta\tau(-FVX - FVY - PRESSV + DIFFV)$$



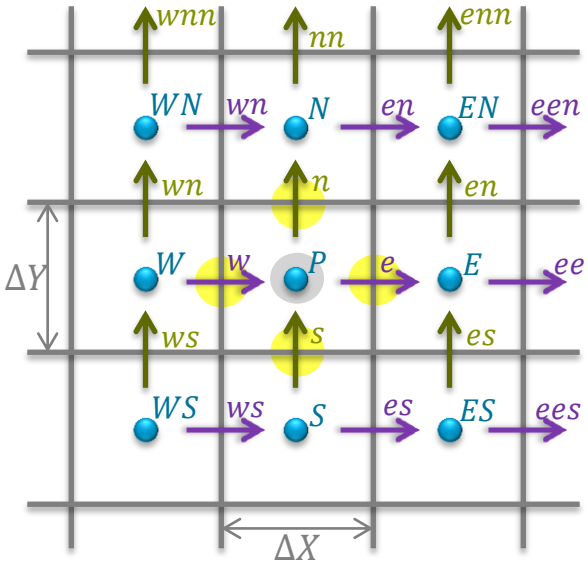

```
__kernel void ocl_hsmac_momentum_v(
  __global float* uo, // velocity-x old
  __global float* vn, // velocity-y new
  __global float* vo, // velocity-y old
  __global float* p,  // pressure
  __global int* vg,   // boundaries
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int x = get_global_id(0);
  unsigned int y = get_global_id(1);
  unsigned int i = x+y*NX2;

  if (x<NX2 && y<NY2 && vg[i]==0)
  {
    float deltaTime = DELTA_TIME;

    float deltaX = DELTA_X;
    float deltaY = DELTA_Y;

    float vdxdx = (((vo[i+1]-vo[ i ])/deltaX)-
                   ((vo[ i ]-vo[i-1])/deltaX)
                   )/deltaX;

    float vdydy = (((vo[i+NX2]-vo[i])/deltaY)-
                   ((vo[i]-vo[ i-NX2])/deltaY)
                   )/deltaY;

    float DIFFV = BB*(vdxdx+vdydy);

    float PRESSV = AA*(p[i+NX2]-p[i])/deltaY;

    float FVX = ((vo[i+1]-vo[i-1])/(2.0f*deltaX))*
    ((uo[i-1+NX2]+uo[i+NX2]+uo[i-1]+uo[i])/4.0f);

    float FVY =
    vo[i]*
    ((vo[i+NX2]-vo[i-NX2])/(2.0f*deltaY));

    vn[i] = vo[i] +
            deltaTime * (-FVX-FVY-PRESSV+DIFFV);
  }
}
```

Pressure and velocity corrections

```
__kernel void ocl_hsmac_press_velocity_correction(
  __global float* un,    // velocity-x
  __global float* vn,    // velocity-y
  __global float* p,     // pressure
  __global int* pg,      // boundaries
  __global float* dmax, // convergence parameter
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int i = get_global_id(0);

  float deltaTime = DELTA_TIME;

  if (i<NX2*NY2 && pg[i]==0)
  {
    float deltaX = DELTA_X;
    float deltaY = DELTA_Y;

    float D =
    (un[i]-un[i-1])/deltaX+
    (vn[i]-vn[i-NX2])/deltaY;

    float DEL =
    deltaTime * ((2.0f/(deltaX*deltaX))+
                 (2.0f/(deltaY*deltaY))
                );

    float CORRP =-D/DEL;

    // correct pressure
    p[i] += OMEGA*CORRP;

    // correct velocity
    un[  i  ] += (deltaTime/deltaX)*CORRP;
    un[ i-1 ] -= (deltaTime/deltaX)*CORRP;
    vn[  i  ] += (deltaTime/deltaY)*CORRP;
    vn[i-NX2] -= (deltaTime/deltaY)*CORRP;

    dmax[0] += fabs(D);
  }
}
```

$$ {}^{m}D_{i,j}^{k+1} = \frac{{}^{m}U_{i,j}^{k+1} - {}^{m}U_{i-1,j}^{k+1}}{\Delta X} + \frac{{}^{m}V_{i,j}^{k+1} - {}^{m}V_{i,j-1}^{k+1}}{\Delta Y} = D $$

$$ {}^{m}\left(\frac{\partial D_{i,j}}{\partial P_{i,j}}\right)^{k+1} = \Delta\tau\left\{\frac{1}{\Delta X}\left(\frac{1}{\Delta X}+\frac{1}{\Delta X}\right)+\frac{1}{\Delta Y}\left(\frac{1}{\Delta Y}+\frac{1}{\Delta Y}\right)\right\} = DEL $$



$$ U'_e = \frac{\Delta\tau}{\Delta X}P'_{i,j} \qquad U'_w = -\frac{\Delta\tau}{\Delta X}P'_{i,j} $$

$$ V'_n = \frac{\Delta\tau}{\Delta Y}P'_{i,j} \qquad V'_s = -\frac{\Delta\tau}{\Delta Y}P'_{i,j} $$

$$ P'_P = -\frac{D}{DEL} $$

Advance flow tables

- Advance velocity:

```
__kernel void ocl_hsmac_advance_u2(
  __global float* un, // velocity-x new
  __global float* uo, // velocity-x old
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int i = get_global_id(0);
  unsigned int j = get_global_id(1);

  unsigned int k = i+j*NX2;

  if (i<NX2 && j<NY2)
  {
    uo[k] = un[k];
  }
}
```

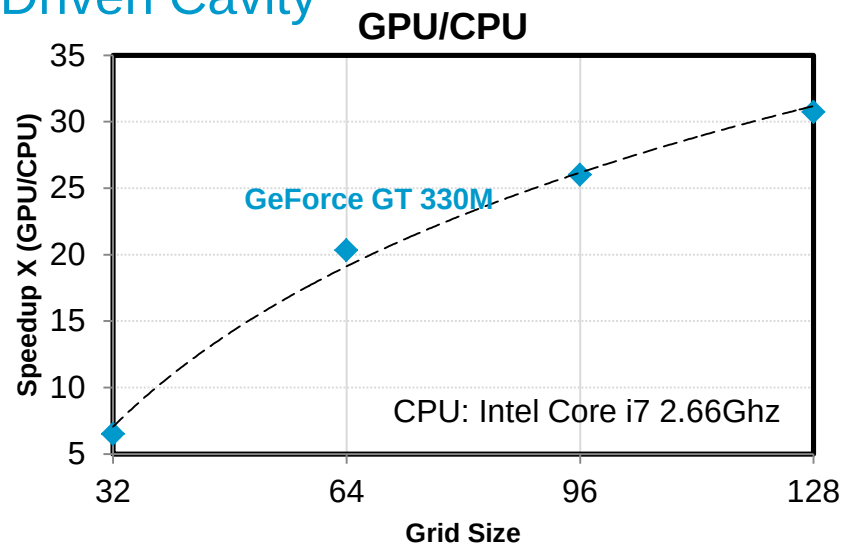
```
__kernel void ocl_hsmac_advance_v2(
  __global float* vn, // velocity-y new
  __global float* vo, // velocity-y old
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int i = get_global_id(0);
  unsigned int j = get_global_id(1);

  unsigned int k = i+j*NX2;

  if (i<NX2 && j<NY2)
  {
    vo[k] = vn[k];
  }
}
```

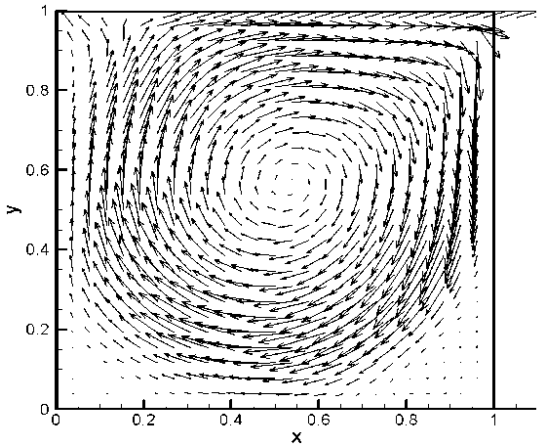
- And for instance, this can be optimised explicitly:

```
__kernel void ocl_hsmac_advance_u2_shared(
  __global float* un,
  __global float* uo,
  const unsigned int NX2,
  const unsigned int NY2,
  __local float* block)
{
  unsigned int x = get_global_id(0);
  unsigned int y = get_global_id(1);

  unsigned int i = x+y*NX2;

  if (x<NX2 && y<NY2)
  {
    block[get_local_id(0)+get_local_id(1)*BLOCK_DIM] = un[i];
  }

  barrier(CLK_LOCAL_MEM_FENCE);

  if (x<NX2 && y<NY2)
  {
    uo[i] = block[get_local_id(0)+get_local_id(1)*BLOCK_DIM];
  }
}
```

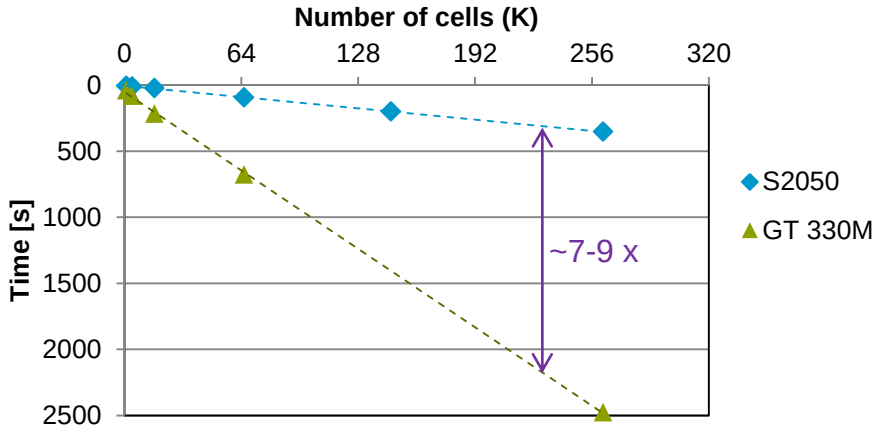
Measures for Execution Speedup

- Driven Cavity





- Natural Convection

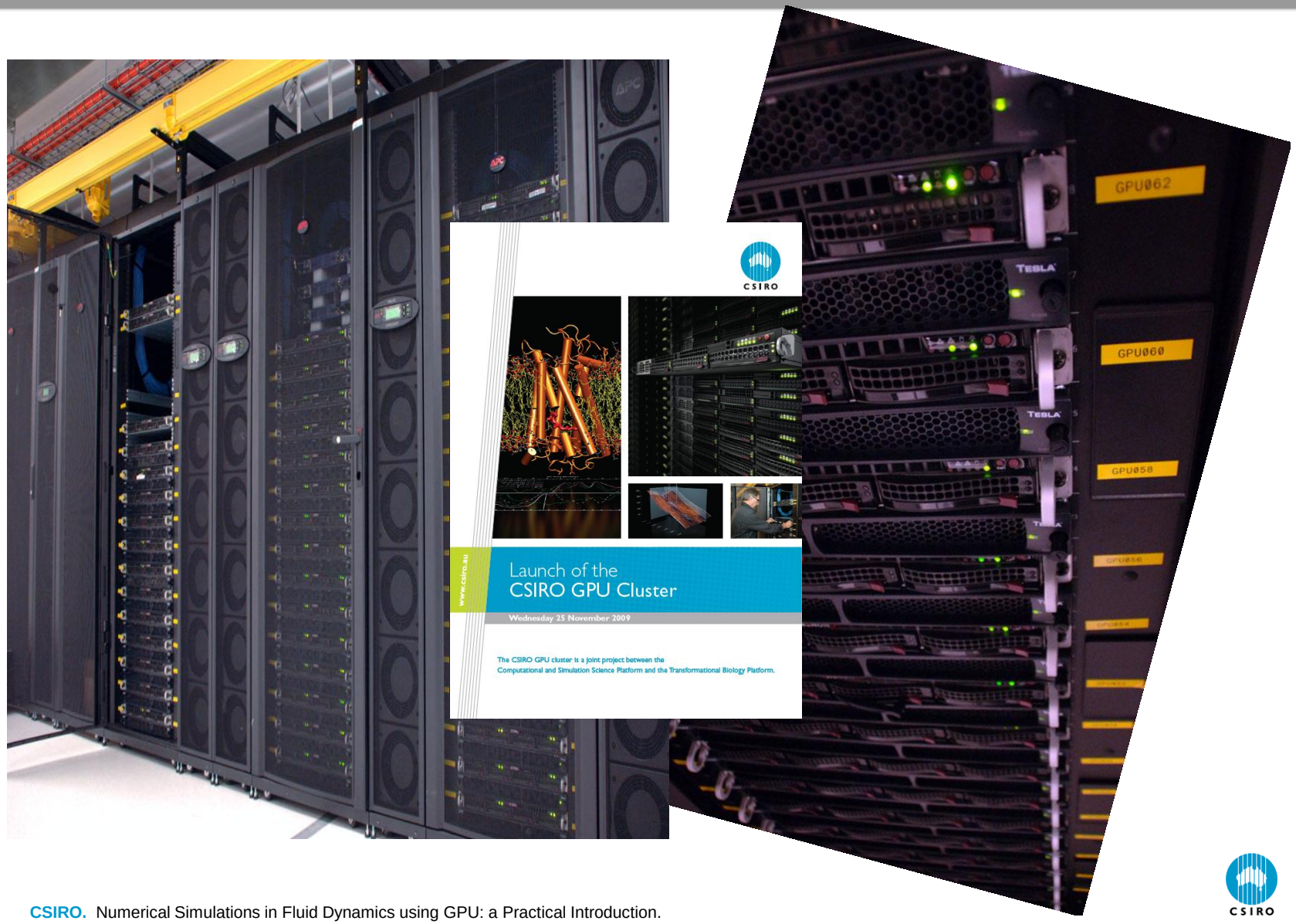


Execution times:

Grid size	GT 130M	S2050
32 x 32	40.64 s	8.17 s
64 x 64	83.54 s	11.37 s
128 x 128	218.27 s	24.14 s
256 x 256	678.58 s	93.61 s
512 x 512	2477.87 s	353.72 s

CSIRO GPU Cluster

CSIRO GPU Cluster Launch (25 November 2009)



CSIRO

CSIRO GPU Cluster Configuration

GPU Cluster Configuration

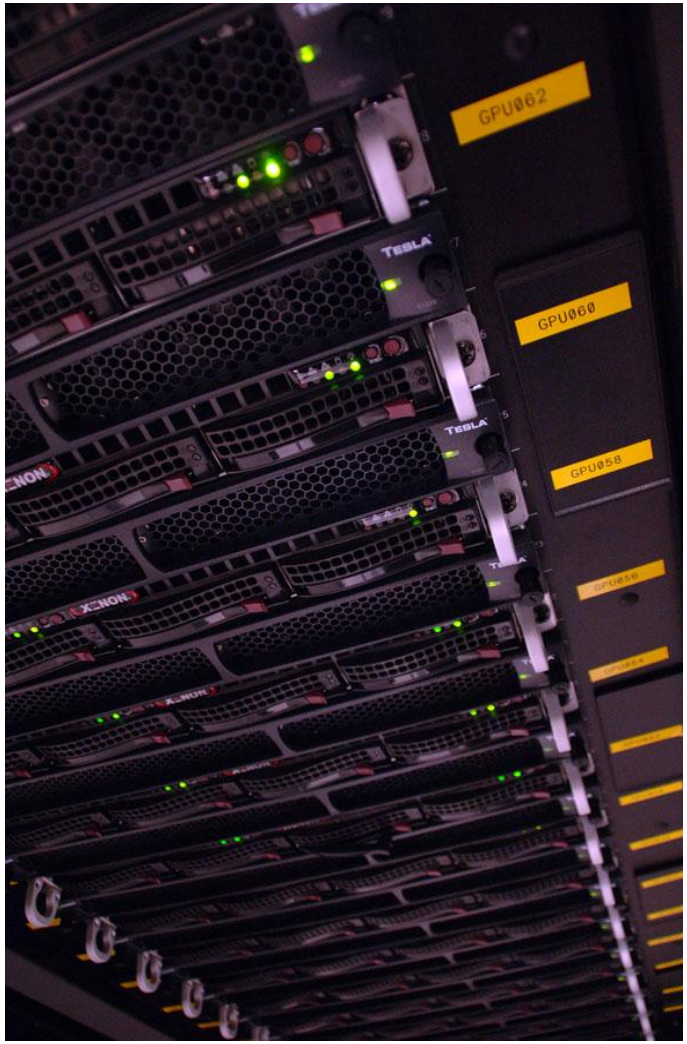
The new CSIRO GPU cluster will deliver up to 264 plus Teraflops of single precision computing performance and will consist of:

•128 Dual Xeon E5462 Compute Nodes (i.e. a total of 1024 2.8GHz compute cores) with 16 or 32GB of RAM, 500GB SATA storage and DDR InfiniBand interconnect

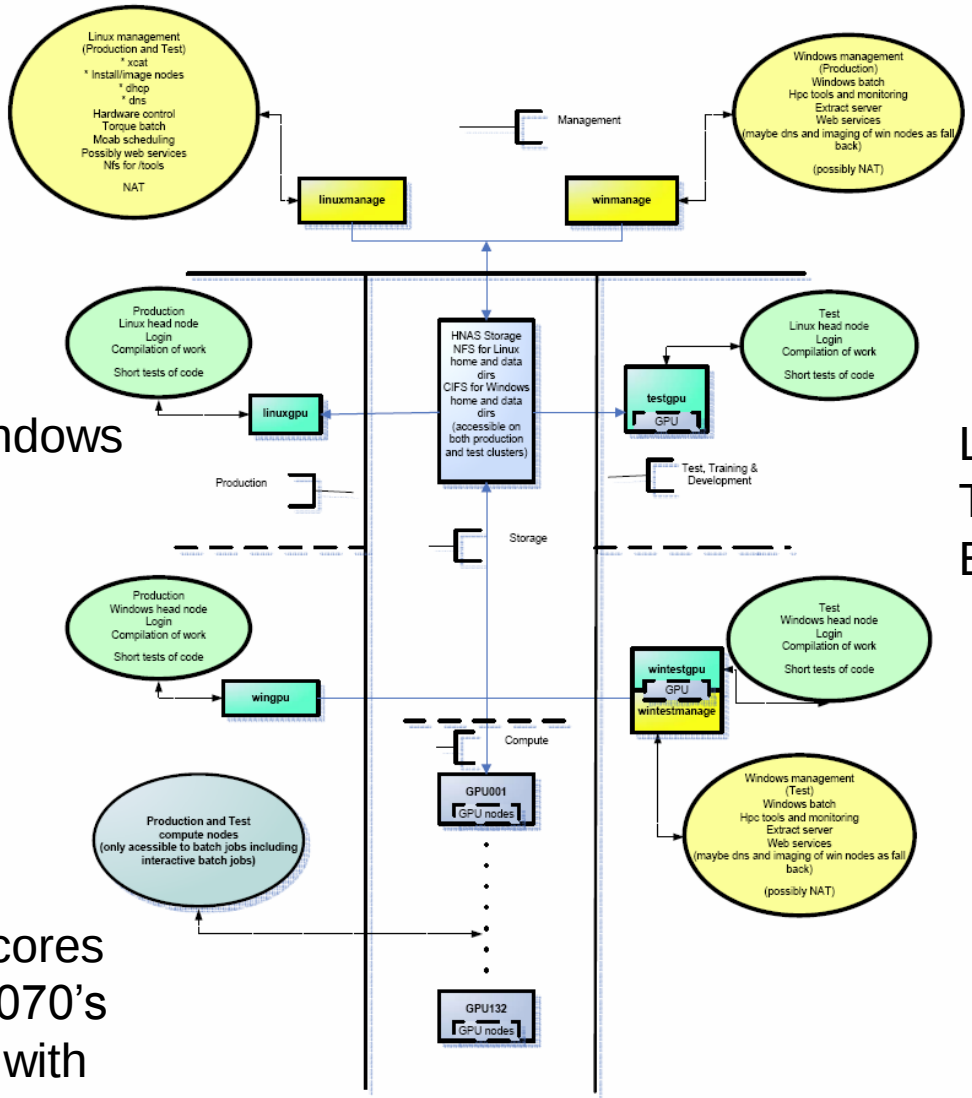
•64 Tesla S250's (256 GPUs with a total of 114,688 CUDA processor cores)

•144 port DDR InfiniBand Switch

•80 Terabyte Hitachi NAS file system.



CSIRO GPU Cluster



Linux and Windows Management Nodes

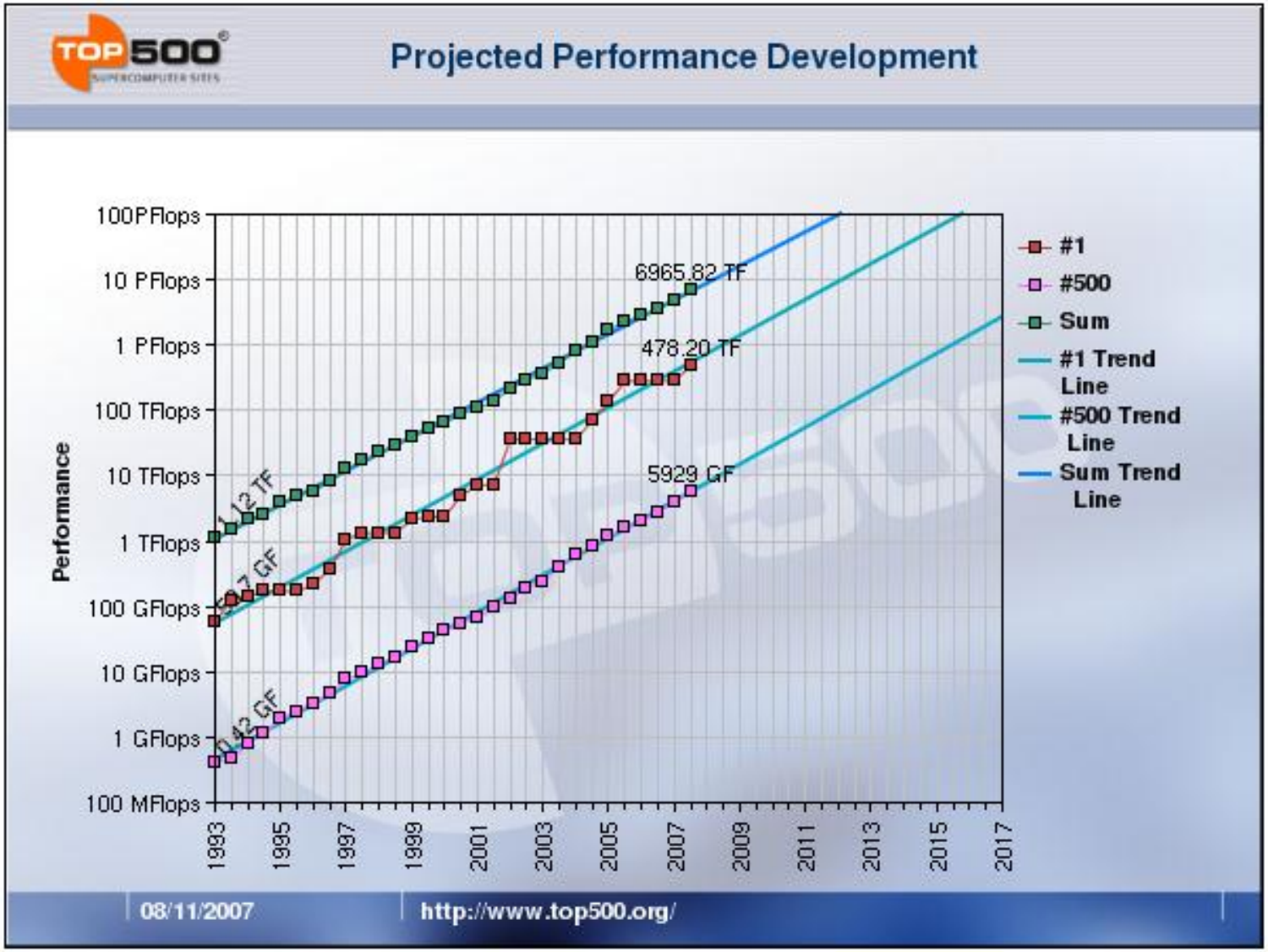
Linux and Windows Production Environment

Linux and Windows Test & Training Environment

Currently
1024 CPU cores
64 Tesla s1070's
(256 GPUs with
61,440 CUDA cores)

CSIRO

CSIRO GPU Cluster



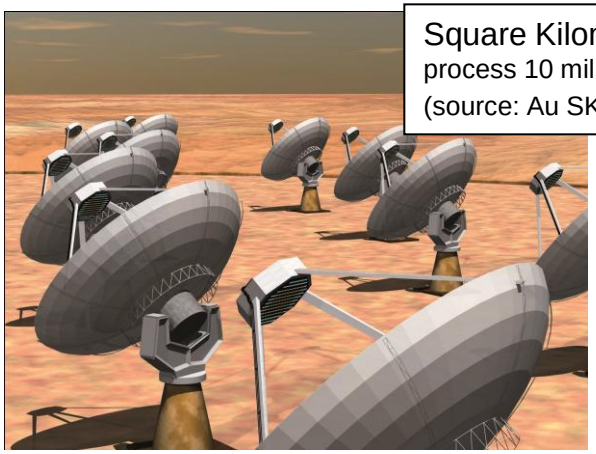
NVIDIA CUDA Research Center

- CSIRO, in recognition of its leadership in massively parallel computing, is an inaugural member of the NVIDIA CUDA Research Center program

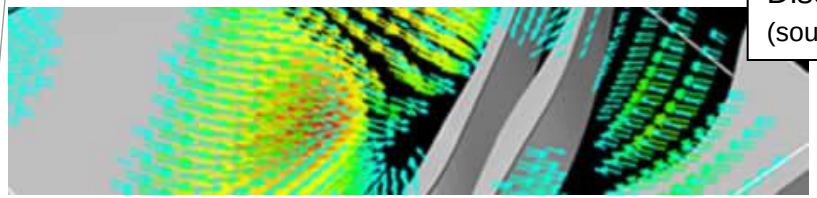
- Institutions identified as CUDA Research Centers are doing world-changing research by leveraging CUDA and NVIDIA GPUs. CUDA Research Centers and PIs:

  - CSIRO (PI: John Taylor)
  - ICHEC (PI: Gilles Civario)
  - Johns Hopkins University (PI: Alex Szalay)
  - Mass General-Harvard Medical School (PI: Homer Pien)
  - Nanyang University (PI: Bertil Schmidt)
  - SINTEF (PI: Trond Runar Hagan)
  - Technical University of Ostrava (PI: Eduard Sojka)

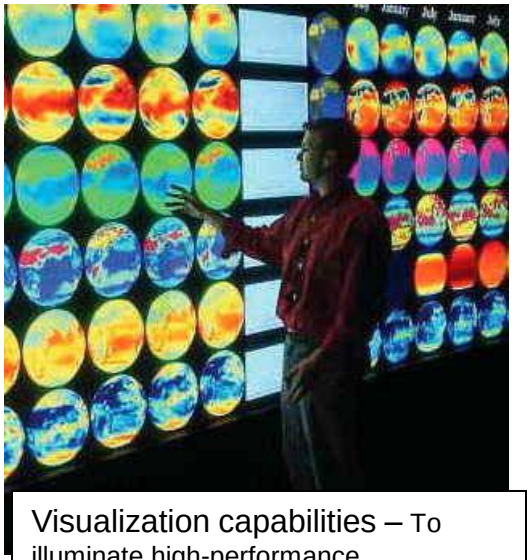
Science: Emerging Science Challenges



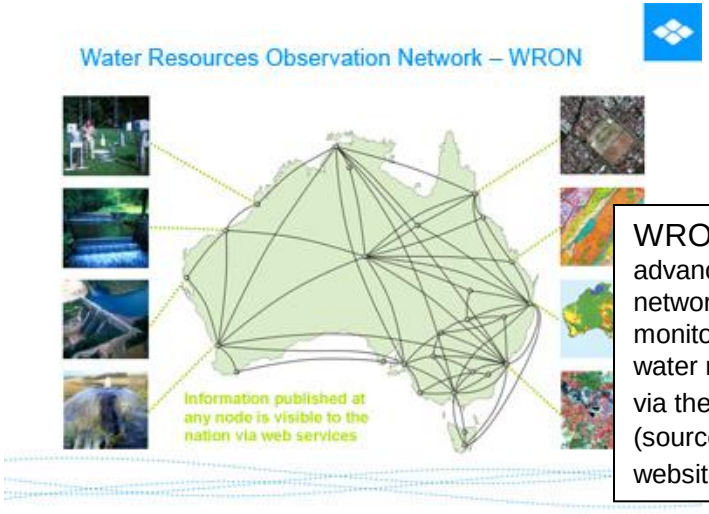
Square Kilometre Array – Potential to process 10 million Gb of data per hour (source: Au SKA website)



Predictive Mineral Discovery – (source:CSIRO E&M)

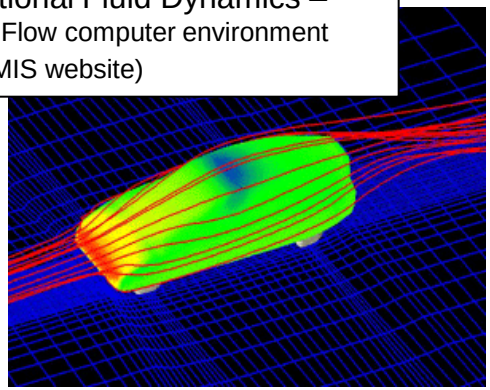


Visualization capabilities – To illuminate high-performance computing data from large-scale climate simulations. (source: US Department of Energy)



WRON – Using advanced sensor networks to monitor scarce water resources via the internet (source: WRON website)

Computational Fluid Dynamics – CMIS Fast Flow computer environment (source: CMIS website)



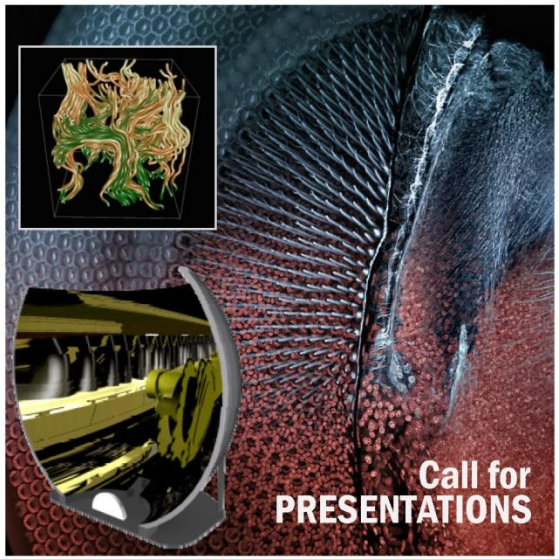
Summary

- CFD does not need to be scary !!!
- We used the following approach:
  - Equations → Discretisation → Implementation → Verification → Results → Transfer CPU code to GPU code → Verification → Results.
- Methods presented:
  - Finite Difference Method (for discretisation),
  - HSMAC Method (for mutual iteration of pressure/velocity fields),
  - BFC (for solving Navier-Stokes equations on complex geometry),
  - Upwind and UTOPIA (for getting more stable and accurate results),
  - OpenCL (for speeding-up computing process).
- The methodology presented, can be extended into 3D and applied to compute different cases in real-time, e.g., smoke, free surface flows, ventilation, turbulent flows, etc.
- Source-code to be released online.

We cordially invite you to participate in **OzViz 2010 workshop** which will be held on 1-3 December 2010 in sunny subtropical **Brisbane**, Queensland. The workshop will bring together scientific visualisation practitioners from Australia and New Zealand in order to share their ideas and work in an informal atmosphere.

This year's **OzViz** program will also include an informative one day **OpenCL workshop** with invited speakers and live demonstrations.

Registration details will be announced soon.



OzViz 2010 workshop

http://sites.google.com/site/ozvizworkshop/ozviz-2010

Call for participation:

We are ready to accept your talk/poster presentation proposals. Please send your proposals to ozviz2010@gmail.com with the subject "***OzViz2010 Proposal***" listing your name, names of co-authors and associated institutions.

Program Committee:

Con Caris, CSIRO Earth Science and Resource Engineering
Drew Whitehouse, Australian National University
Paul Bourke, University of Western Australia
Tomasz Bednarz, CSIRO Earth Science and Resource Engineering
Derek Gerstmann, University of Western Australia

Important dates & locations:

1st November: deadline for presentation abstract submission,
7th November: acceptance notification,
1st December: OpenCL workshop (*Queensland Centre for Advanced Technologies*)
2nd -3rd December: Presentations and posters (*University of Queensland*)

Publication types and formats:

This year, **OzViz** would like to offer participants the opportunity to submit either a standard paper or a technical paper that outlines details of a specific issue/problem and your innovative solution that might assist others. See last year's topics at:
http://sites.google.com/site/ozvizworkshop/ozviz2009-call-for-registration-1

Paper and abstract templates are available for download on the **OzViz** website. Participants are free to prepare and propose one of the following:

(1) Oral presentation + short abstract
(2) Oral presentation + short paper (max 3 pages A4)
(3) Poster + short abstract or paper

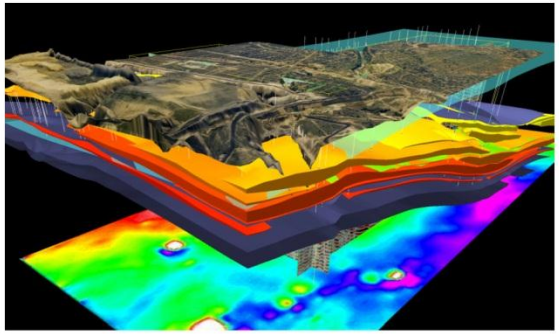
Submissions should include text and images in PDF format.

Cost:

OzViz 2010 and OpenCL workshops will be free for all participants.

Registration:

Registration details will be made available closer to the abstract submission date.



OpenCL workshop

Introduction:

OpenCL* **(Open Computing Language)** is an open, royalty-free, cross-platform standard specifically designed for general purpose parallel programming of heterogeneous systems, including modern desktop and workstation class multi-core processors (CPUs), graphics processing units (GPUs), and other accelerators such as Cell and digital signal processors (DSPs). The standard includes an API for coordinating execution between devices and a cross-platform parallel programming language. Based on ISO C99, the OpenCL-C language provides a well-defined execution environment with strict precision requirements, allowing device agnostic kernel methods to be written which can be run on any supported OpenCL device. The OpenCL standard supports task-based and data-based parallel computing, features consistent numerical requirements based on the IEEE standard for floating-point arithmetic (IEE 754), defines a configuration profile for handheld and embedded devices and efficiently interoperates with OpenGL, OpenGL ES, and other graphics APIs.

Confirmed Presenters:

Derek Gerstmann (UWA), Mark Harris (NVIDIA), Justin Hensley / Jason Yang (AMD), John Taylor (CSIRO), Tomasz Bednarz (CSIRO).

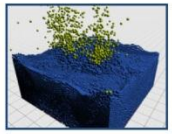
Aim:

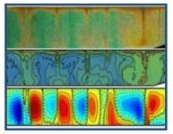
The aim of the OpenCL workshop will be to provide registered attendees with an intensive one day course which covers both a general introduction as well more advanced topics showcasing OpenCL as a technology platform. Speakers from both industry and academia will discuss a range of subjects, including core fundamentals, hardware architectures, parallel programming, as well as workload scheduling and device specific optimisations.

Program of the workshop:

Detailed program of the workshop will be announced soon with the topic covering: Introduction to OpenCL and specification overview, OpenCL compared to other technologies, Parallel Architectures, OpenCL Memory Model, OpenCL API and Kernel Language, OpenCL C++ bindings, OpenCL/OpenGL interoperability, OpenCL for visualisation, examples, OpenCL and numerical simulations, examples.

The workshop is planned to be an interactive event where attendees will have an opportunity to test most of the presented examples using their own machines (instructions will be sent in late November 2010).





Registration and More Information:

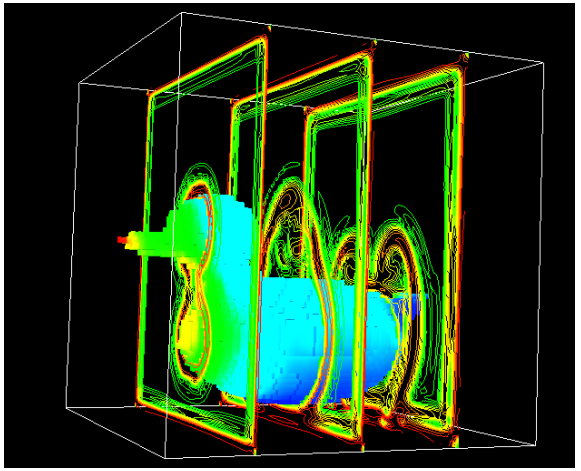
Registration details will be made available closer to the abstract submission date. If you require more information about the OpenCL workshop, please send an e-mail with your question to ozviz2010ocl@gmail.com.

*OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

CSIRO

Earth Science and Resource Engineering
Dr Tomasz P Bednarz
3D Visualisation Engineer
Mining Automation

Phone: +61 429 153 274
Email: tomasz.bednarz@csiro.au
Web: www.csiro.au/org/CESRE.html
www.tomaszbednarz.com



Acknowledgments:
Dr J.Taylor, C.Caris, Prof. H.Ozoe, Prof. H.Hirano, Prof. T.Tagawa, Prof. J.C.Patterson, Prof. Chengwang Lei, Prof. J.S.Szmyd, Dr P.Cleary, Dr M.Rudman, Dr J.Ralston, Dr L.Domanski, Dr J.Malos, J.Craig, Dr S.Zawlodzka-Bednarz

www.csiro.au

Thank you… …

Contact Us
Phone: 1300 363 400 or +61 3 9545 2176
Email: enquiries@csiro.au Web: www.csiro.au

CSIRO