Department of Hydrodynamic Systems

MPGOS: A modular and general-purpose program package to solve a large number of independent ODE systems

Ferenc **Hegedűs**

Budapest University of Technology and Economics, Department of Hydrodynamic Systems, Budapest, Hungary

H-1111, Budapest, Műegyetem rkp. 3. D building. 3 rd floor
Tel: 00 36 1 463 16 80 Fax: 00 36 1 463 30 91
www.hds.bme.hu

MŰEGYETEM 1782

MPC in ODEs

- *How to apply massively parallel computations in IVP?*
- *Large number of instances of independent and identical ODE systems*
  - ***Parameter studies (e.g. bifurcation analysis)***
  - ***Different initial conditions (e.g. multistability)***

The From of the System

- *An instance of ODE:* $\dot{\underline{x}} = f\left(\underline{x}, t; \underline{p}\right) \qquad t \in (t_0, t_1)$

- $\underline{x}$ *is the vector of the state variables*
- t *is the time*
- *The dot stands for the derivative with respect to time*
- $\underline{p}$ *is the vector of parameters having 3 subcategories:*
  - ***Control Parameters*** *(different from instance to instance)*
  - ***Shared Parameters*** *(common to all instances)*
  - ***Accessories*** *(user programmable parameters)*

Overview of the Capabilities

- *Framework in C++ to completely hides GPU programming*
- *Simple syntax*

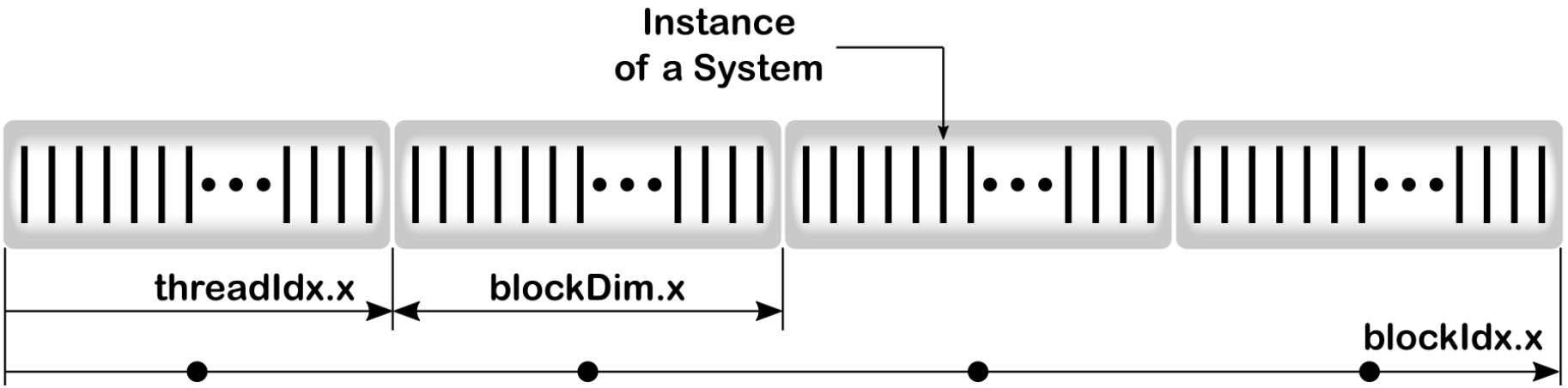
- *Only explicit solvers:*
  - *4^{th} order Runge—Kutta*
  - *Adaptive Runge—Kutta—Cash—Karp of orders 4 and 5*

- *No dense output*

- ***Event handling***
- ***User interaction of after every successful time step and/or event detection***
- ***Easy usage of multi-GPUs***

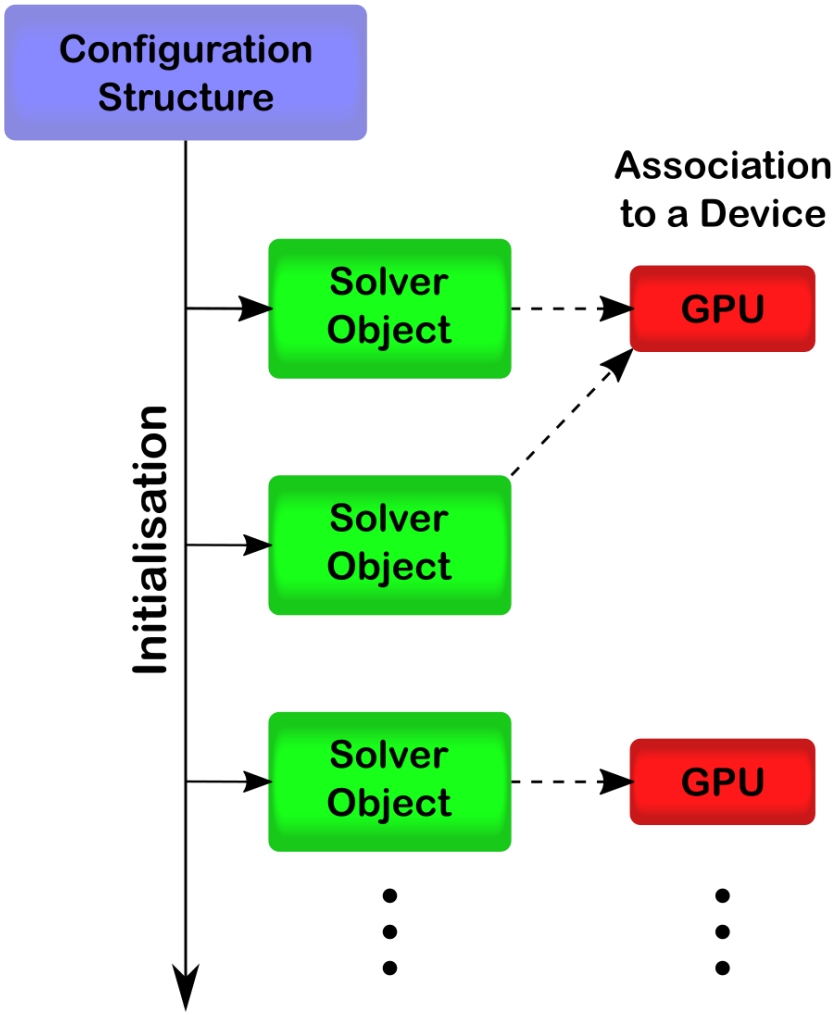
Implementation Details

- *The parallelisation strategy*



Implementation Details

- *Configuration of the **Solver Object***



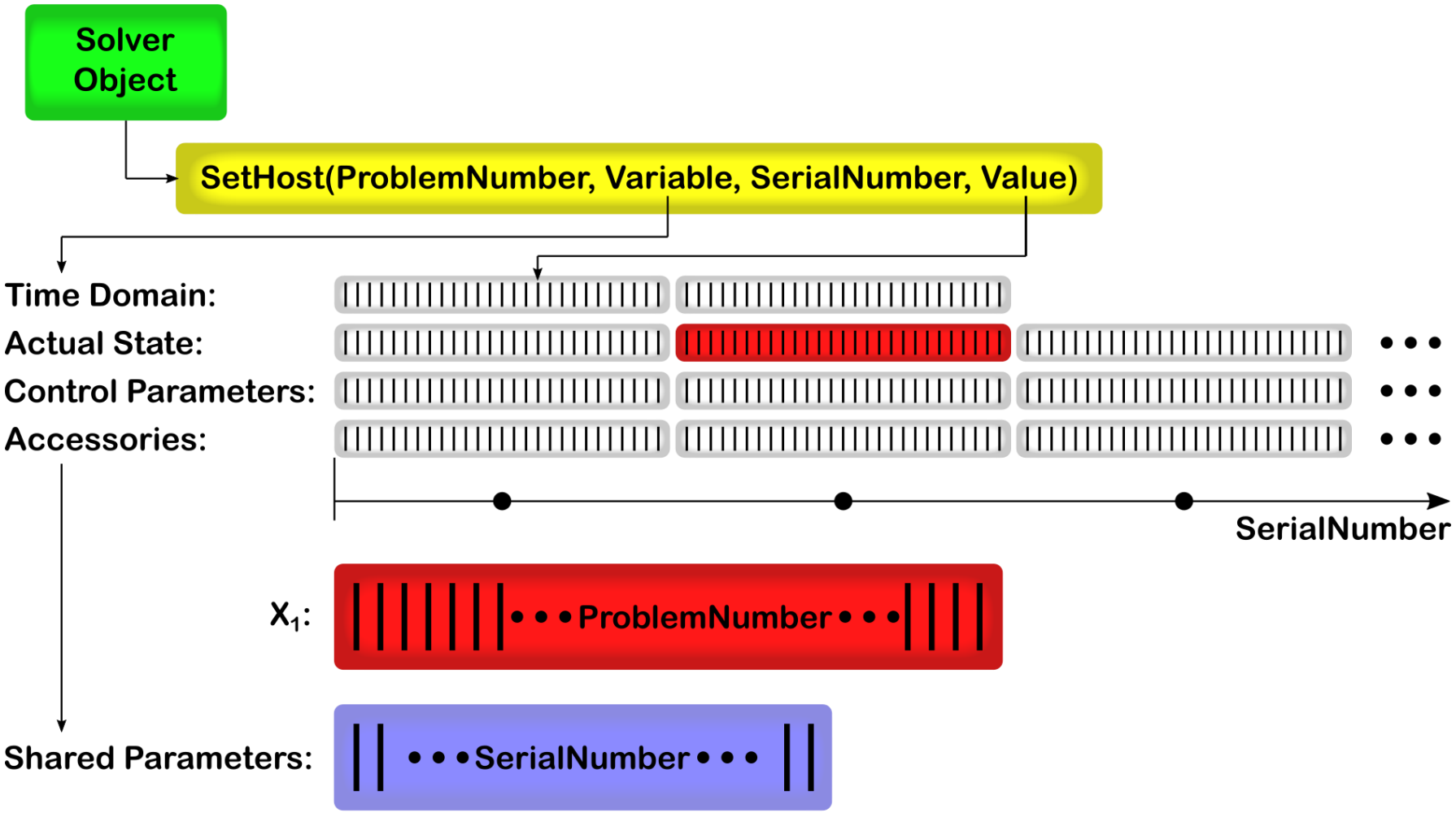
Implementation Details

- *Configuration of the **Solver Object***

```
int NumberOfThreads = 46080;


ConstructorConfiguration Configuration;
    Configuration.NumberOfThreads            = NumberOfThreads;
    Configuration.SystemDimension            = 2;
    Configuration.NumberOfControlParameters  = 1;
    Configuration.NumberOfSharedParameters   = 1;
    Configuration.NumberOfEvents             = 0;
    Configuration.NumberOfAccessories        = 0;


ProblemSolver Solver1(Configuration, DeviceNumber1);
ProblemSolver Solver2(Configuration, DeviceNumber1);
```

Implementation Details

- *Fill-up the **Solver Object** with data*



Implementation Details

- *Fill-up the **Solver Object** with data*

```
int ProblemNumber = 0;

for (int k=0; k<NumberOfThreads; k++)
{
    Solver1.SetHost(ProblemNumber, TimeDomain,  0, 0 );
    Solver1.SetHost(ProblemNumber, TimeDomain,  1, 2*PI );

    Solver1.SetHost(ProblemNumber, ActualState, 0, -0.5);
    Solver1.SetHost(ProblemNumber, ActualState, 1, -0.1 );

    Solver1.SetHost(ProblemNumber, ControlParameters, 0, p1[k]);

    ProblemNumber++;
}

Solver1.SetHost(SharedParameters, 0, 0.3 );
Solver1.SetHost(SharedParameters, 1, 5.6 );
```

Implementation Details

- *The right-hand side, a dedicated device function*

```
__device__ void PerThread_OdeFunction(int tid, int NT, double* F,
double* X, double T, double* cPAR, double* sPAR, double* ACC)
{
    F(0) = X(1);
    F(1) = X(0) - X(0)*X(0)*X(0) - cPAR(0)*X(1) + sPAR(0)*cos(T);
}
```

Singe-well Duffing oscillator

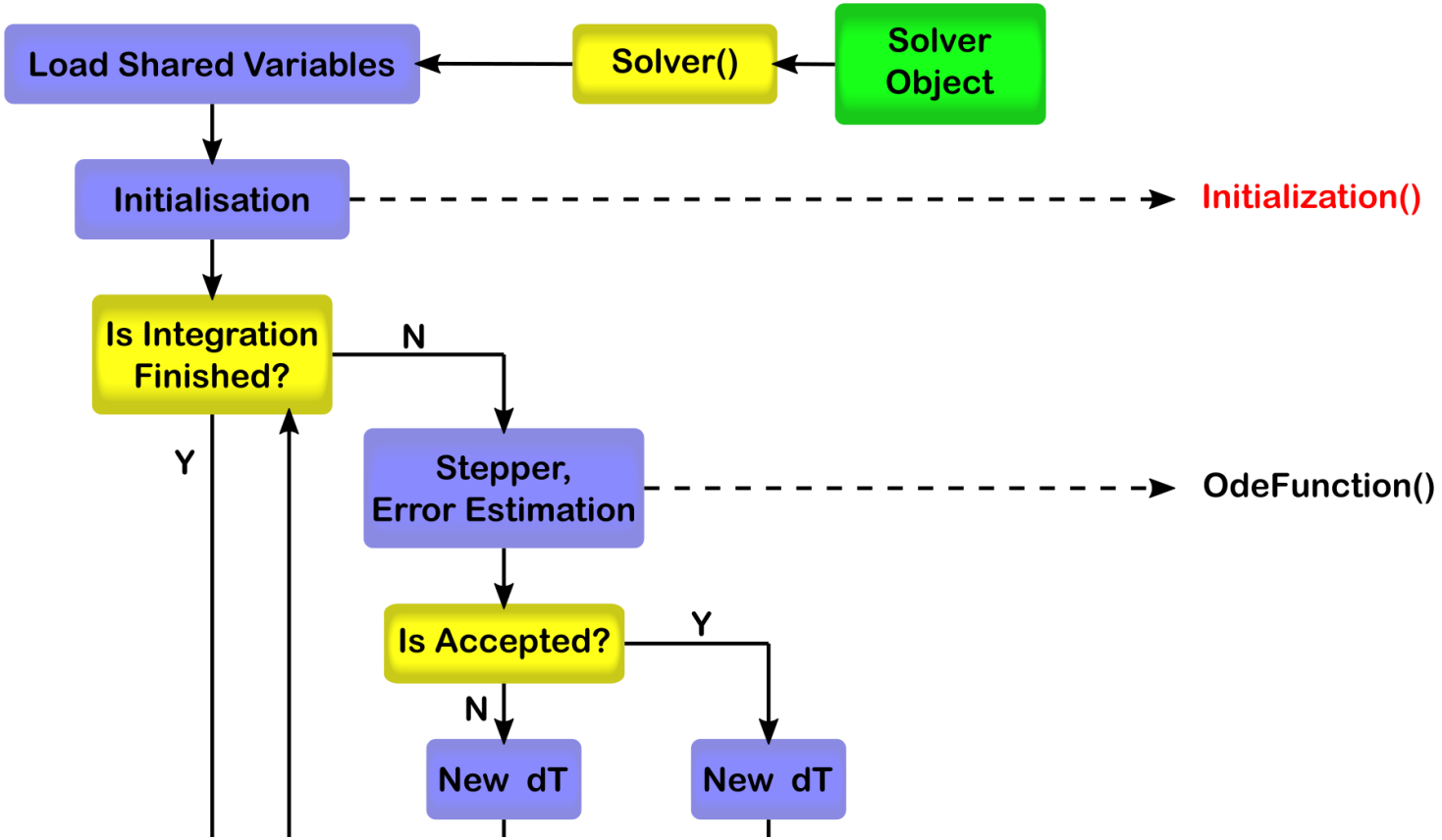
Implementation Details

- *Perform integration*

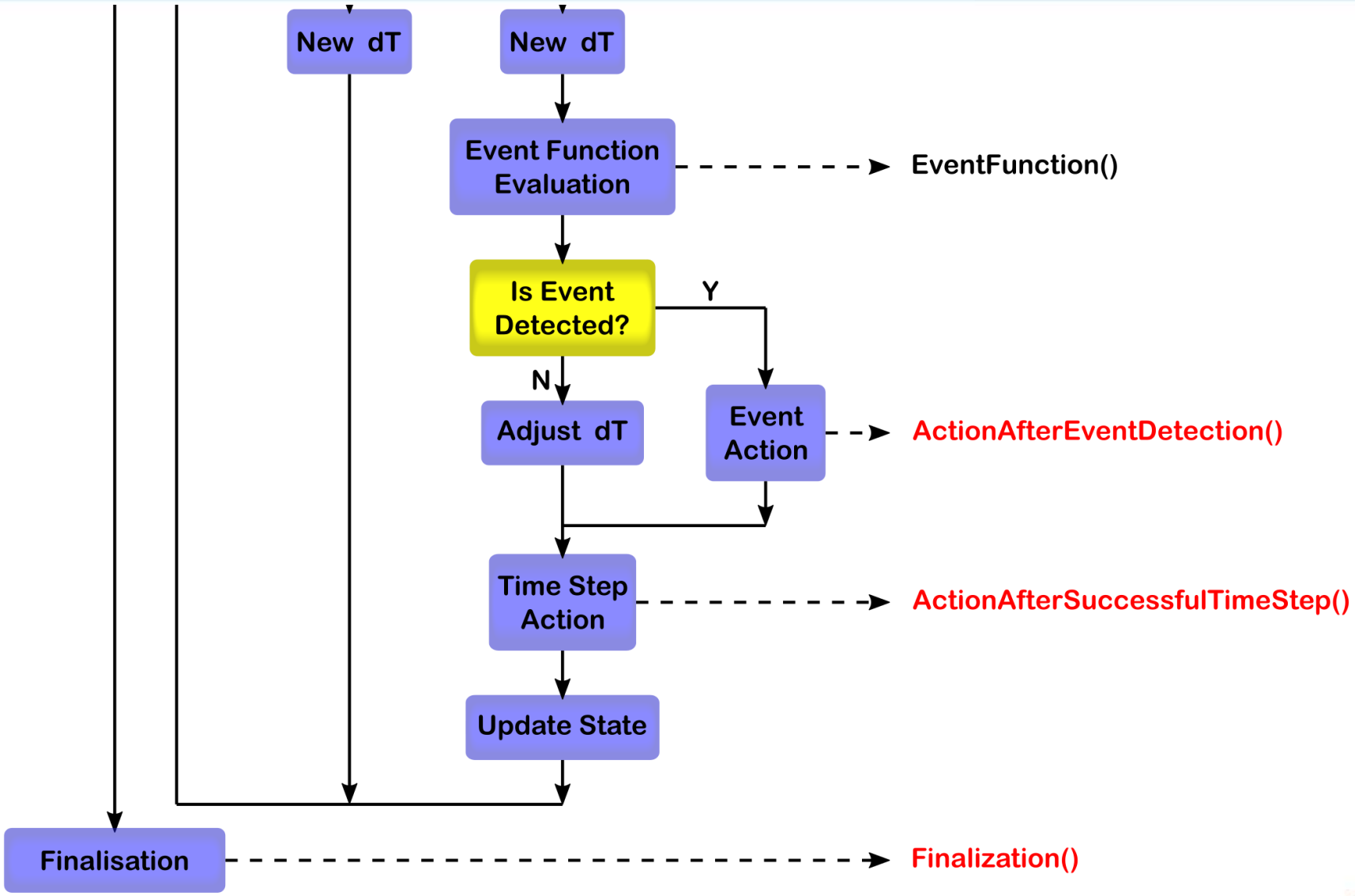
```
SolverConfiguration SolverConfigurationSystem;
    SolverConfigurationSystem.BlockSize       = BlockSize;
    SolverConfigurationSystem.InitialTimeStep = 1e-2;
    SolverConfigurationSystem.Solver          = RKCK45;
    SolverConfigurationSystem.ActiveThreads   = NumberOfThreads;


// INITIAL TRANSIENT
for (int i=0; i<1024; i++)
    Solver1.Solve(SolverConfigurationSystem);


// CONVERGED ITERATIONS
for (int i=0; i<32; i++)
{
    Solver1.Solve(SolverConfigurationSystem);

    SaveData(Solver1, DataFile, NumberOfThreads);
}
```

Implementation Details

- *Inside the **Solver** member function*



Implementation Details



Reference Case I

- *The Duffing oscillator:*

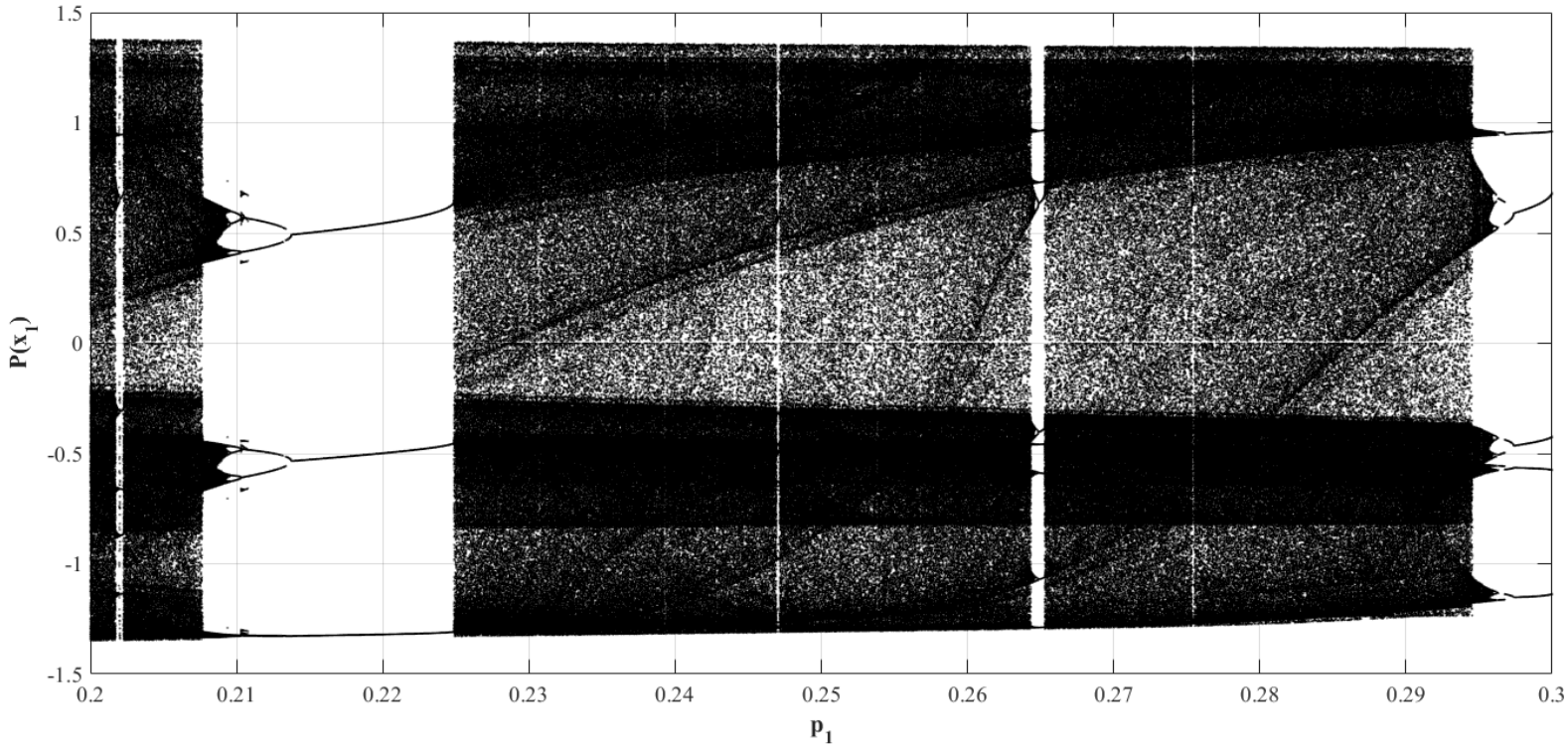
$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = x_1 - x_1^3 - p_1 x_2 + p_2 \cos t$$

- *Parameters:*
  - *p1: damping parameter (control parameter),* ***varied between 0.2 and 0.3 resolution is 46 080***
  - *p2: excitation amplitude*

- *The state space is periodic in time:* $t \in (0, 2\pi)$

Reference Case I

- *Results: 1D bifurcation diagram*



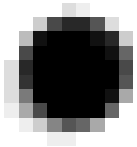
Total number of parameters: 46 080

2 X Nvidia Tesla K20m: 27 s (incl. 1024 transients)

Reference Case II

- *Keller—Miksis equation (collapse of a dual-frequency driven gas bubble):*

$$\left(1-\frac{\dot{R}}{c_L}\right)R\ddot{R}+\left(1-\frac{\dot{R}}{3c_L}\right)\frac{3}{2}\dot{R}^2=\left(1+\frac{\dot{R}}{c_L}+\frac{R}{c_L\rho_L}\frac{d}{dt}\right)\frac{p_L-P_\infty-p_\infty(t)}{\rho_L}$$



- *Dual-frequency excitation:*

$$p_\infty(t)=P_{A1}\sin(2\pi f_1\cdot t)+P_{A2}\sin(2\pi f_2\cdot t+\Theta)$$

- *Investigated parameter space:*

$$P_{A1},\ P_{A2},\ f_1,\ f_2,\ \Theta,\ R_E$$

Reference Case II

- *Parameter values and their resolution*

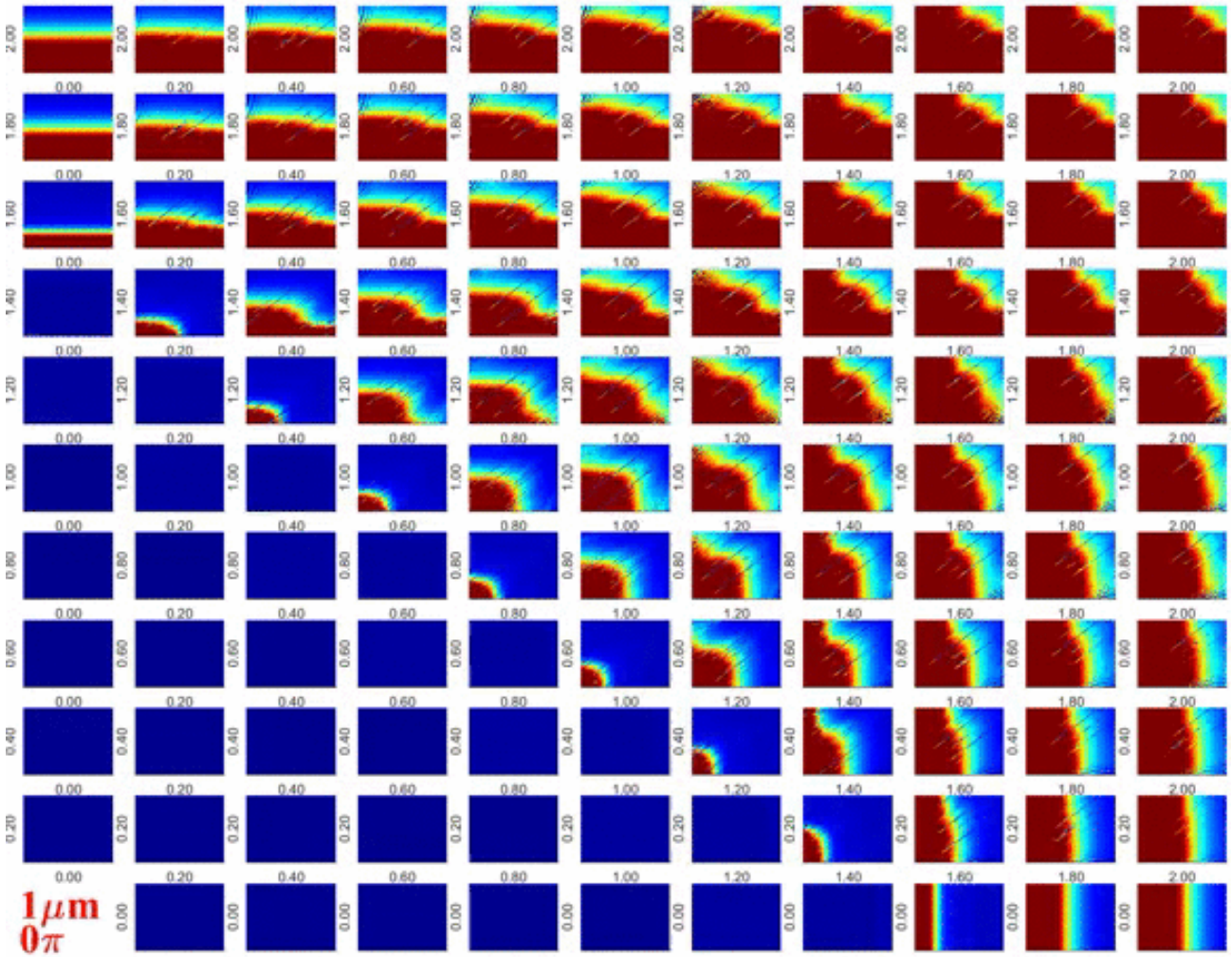
	MIN	MAX	RES.
P_{A1}	0 bar	2 bar	21 (lin)
P_{A2}	0 bar	2 bar	21 (lin)
f_1	20 kHz	2 MHz	101 (log)
f_2	20 kHz	2 MHz	101 (log)
θ	0 rad	1.9π rad	20 (lin)
R_E	1 μm	10 μm	21 (lin)

Total number of parameter combinations:
1 889 429 220 (approx. 2 billion)

Oracle Cloud, 2 X Nvidia Tesla P100: 23 days
(incl. 512 transient iter.)

Reference Case II

- *Collapse strength in a 6D parameter space*



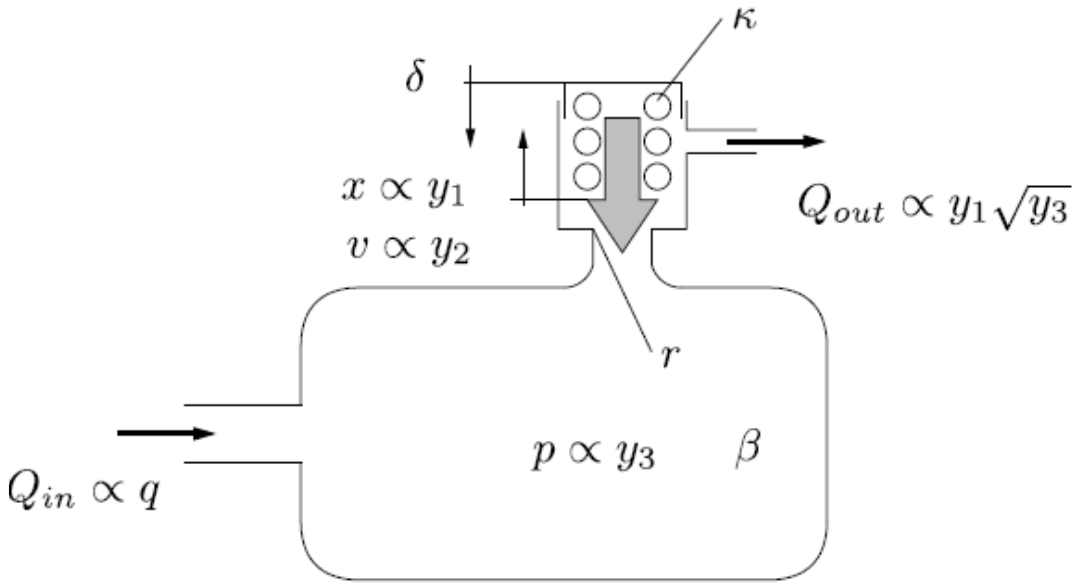
Reference Case III

- *A model of a pressure relief vale (impact dynamics):*

$$\dot{y}_1 = y_2$$
$$\dot{y}_2 = -\kappa y_2 - (y_1 + \delta) + y_3$$
$$\dot{y}_3 = \beta(q - y_1\sqrt{y_3})$$

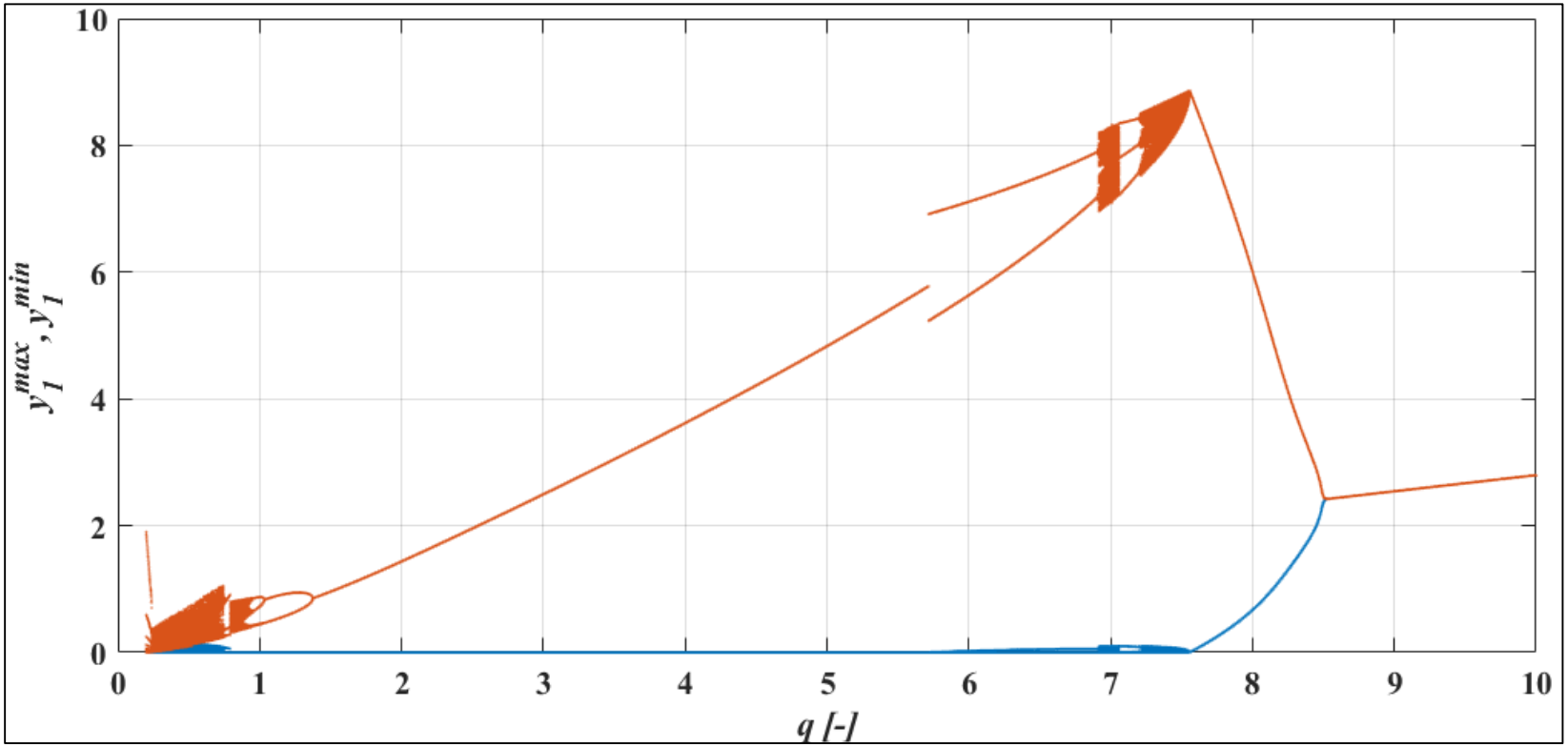
- *The impact law at* $y_1 = 0$*:*

$$y_1^+ = y_1^- = 0$$
$$y_2^+ = -r y_2^-$$
$$y_3^+ = y_3^-$$



Reference Case III

- *A 1D bifurcation diagram with impact:*

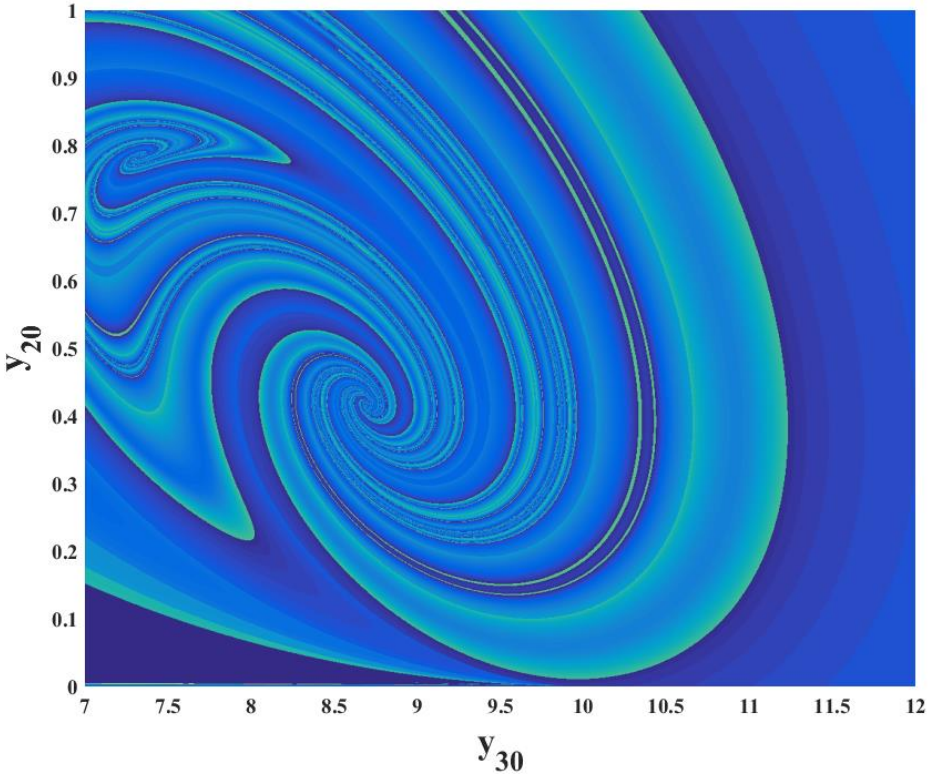


Total number of parameters: 30 720

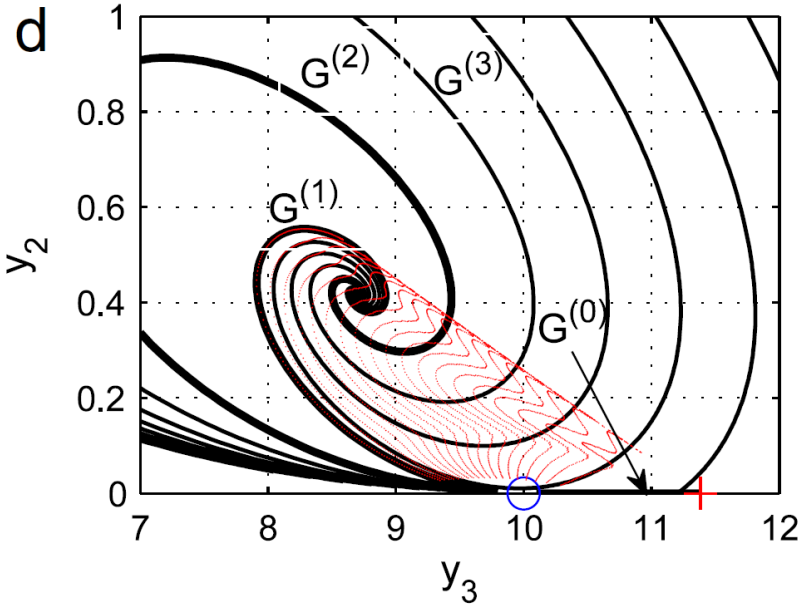
Nvidia Titan Black: 42 s (incl. 1024 transient iter.)

Reference Case III

- *Grazing lines ($y_1 = 0$):*



Hős & Champneys, (2012) Physica D 241 (2012) 2068–2076



Total number of initial conditions: 1024 x 1024

Nvidia Titan Black: 4 s

Summary

- *First version of the code*
  - *User friendly*
  - *Already capable of handling a variety of problem*

- *Further developments:*
  - ***Additional numerical schemes***
  - ***Dense output***
  - ***Delay Differential Equations***
  - ***Implement efficient global/diffusional couplings of identical systems***

www.gpuode.com

Thank you for your attention!

Reference Case II

- *Within the framework of international cooperation*
- *Prof. Dr Werner Lauterborn*
  - *Drittes Physikalisches Institut, Georg-August-Universität Göttingen, Germany*
- *Dr Robert Mettin*
  - *Drittes Physikalisches Institut, Georg-August-Universität Göttingen, Germany*
- *Prof. Dr Ulrich Parlitz*
  - *Research Group Biomedical Physics, Max Planck Institute for Dynamics and Self-Organization, Göttingen, Germany*

Reference Case III

- *Agreement of an international cooperation*

- *Prof. Dr Alan R. Champneys*
  - *Department of Engineering Mathematics, University of Bristol, Bristol*

- *Dr Csaba Hős*
  - *Budapest University of Technology and Economics, Department of Hydrodynamic Systems, Budapest, Hungary*

Details: The Accessories

- *The right hand side:*

```
__device__ void ParametricODE_Solver_OdeFunction(double*
RightHandSide, int idx, int NoT, double t, double*
StateVariable, double* Parameter)
{
        double x1 = StateVariable[idx + 0*NoT];
        double x2 = StateVariable[idx + 1*NoT];

        double p0 = Parameter[idx + 0*NoT];
        double p1 = Parameter[idx + 1*NoT];
        double p2 = Parameter[idx + 2*NoT];
        double p3 = Parameter[idx + 3*NoT];
        double p4 = Parameter[idx + 4*NoT];
        double p5 = Parameter[idx + 5*NoT];
        double p6 = Parameter[idx + 6*NoT];
        double p7 = Parameter[idx + 7*NoT];
        double p8 = Parameter[idx + 8*NoT];
        double p9 = Parameter[idx + 9*NoT];
        double p10 = Parameter[idx + 10*NoT];
        double p11 = Parameter[idx + 11*NoT];
        double p12 = Parameter[idx + 12*NoT];

        double rx1 = 1.0/x1;
        double p = pow(rx1, p10);
        double s1;
        double c1;

        sincospi(2.0*t, &s1, &c1);

        double s2 = sin(2.0*p11*PI*t+p12);
        double c2 = cos(2.0*p11*PI*t+p12);
        double N;
        double D;
        double rD;

        N = (p0+p1*x2)*p - p2*(1.0+p9*x2) - p3*rx1 - p4*x2*rx1
- 1.5*(1.0-p9*x2/3.0)*x2*x2 - ( p5*s1 + p6*s2 ) * (1.0+p9*x2) -
x1*( p7*c1 + p8*c2 );
        D = x1 - p9*x1*x2 + p4*p9;
        rD = 1.0/D;

        RightHandSide[idx + 0*NoT] = x2;
        RightHandSide[idx + 1*NoT] = N*rD;
}
```

The importance of ODEs

- *Why it is important to deal with initial value problems of ODE systems?*

Many physical, biological, economical and social processes can be described by Ordinary Differential Equations

Even Partial Differential Equations are usually decomposed into a large system of ODEs

MPC in ODEs

- *How to apply massively parallel computations in IVP?*
  - *Large ODE system of identical equations*
    - *Discretization of PDE*
    - *Global/diffusional coupling*
  - *Large number of independent, identical ODE systems:*
    - ***Parameter studies (e.g. bifurcation analysis)***
    - ***Different initial conditions (basin of attraction)***

The From of the Systems

- *Systems of ODEs:* $\dot{\underline{x}} = f\left(\underline{x}, t; \underline{p}\right) \qquad t \in (t_0, t_1)$

- $\underline{x}$ *is the vector of the state variables*
- t *is the time*
- *The dot stands for the derivative with respect to time*
- $\underline{p}$ *is the vector of parameters having 3 subcategories:*
  - ***Parameters***
  - ***SharedParameters*** *(cached parameters)*
  - ***Accessories*** *(user programmable parameters)*

Reference Case III

- *The event functions (Poincaré section and impact):*

```
__device__ void PerThread_EventFunction(int tid, int NT, double* EF, double* X, double T, double* cPAR, double*
sPAR, double* ACC)
{
	int i0 = tid + 0*NT;
	int i1 = tid + 1*NT;

	double y1 = X[i0];
	double y2 = X[i1];

	EF[i0] = y2; // Poincaré section
	EF[i1] = y1; // Impact detection
}
```

- *User interaction after every successful event detection:*

```
__device__ void PerThread_ActionAfterEventDetection(int tid, int NT, int IDX, int CNT, double &T, double &dT,
double* TD, double* X, double* cPAR, double* sPAR, double* ACC)
{
	int i1 = tid + 1*NT;

	double p5 = sPAR[3];

	if ( IDX == 1 )
		X[i1] = -p5 * X[i1];
}
```