Department of Hydrodynamic Systems

GPU accelerated solution of 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

The importance of ODE-IVPs

- *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

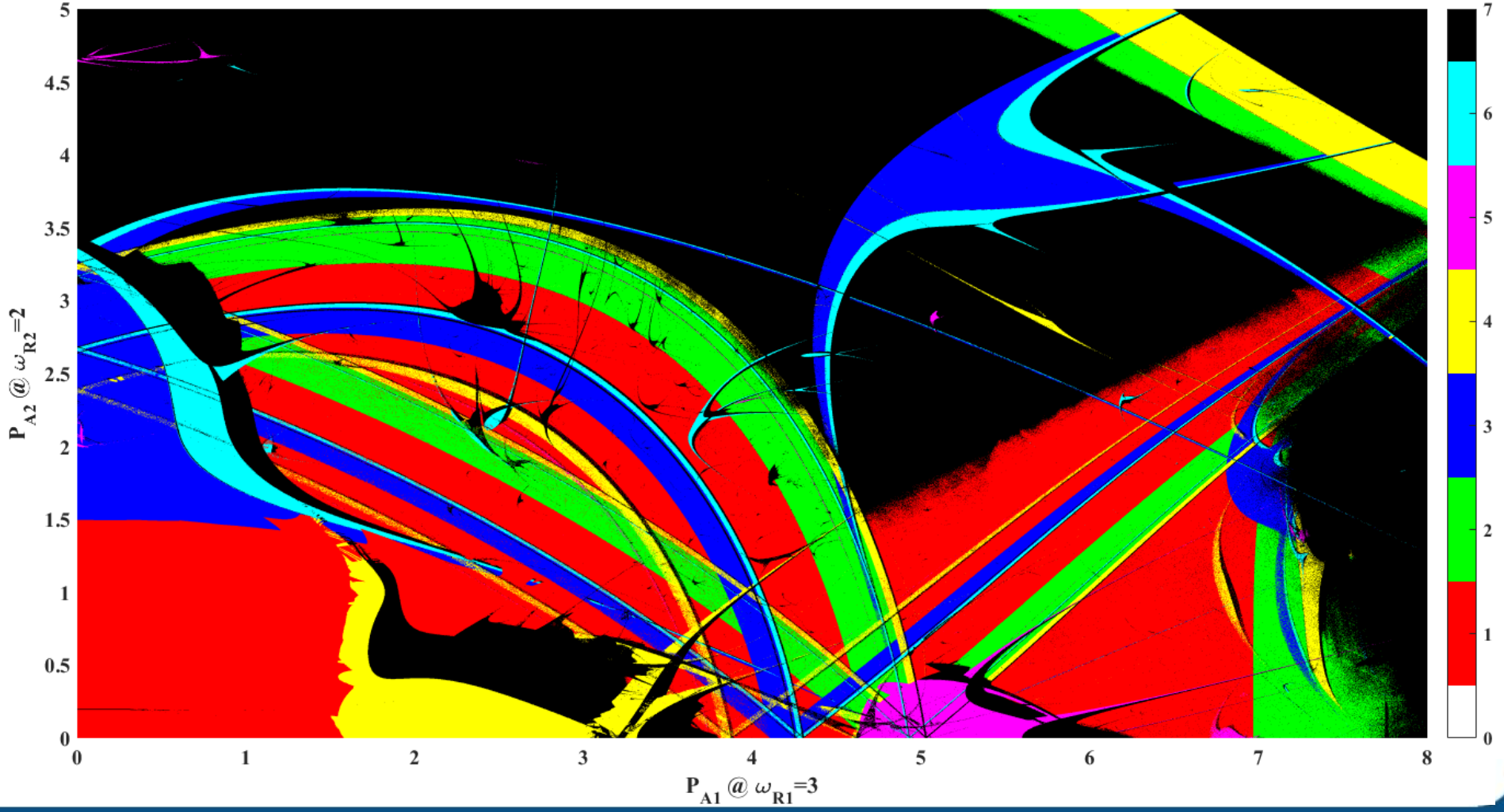
Even Partial Differential Equations are usually decomposed into a large system of ODEs

MPC in ODE-IVPs

- *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)***

Example: Control of Multistab.

- *Novel non-feedback technique to control multistability:*



Solver: Main Aims

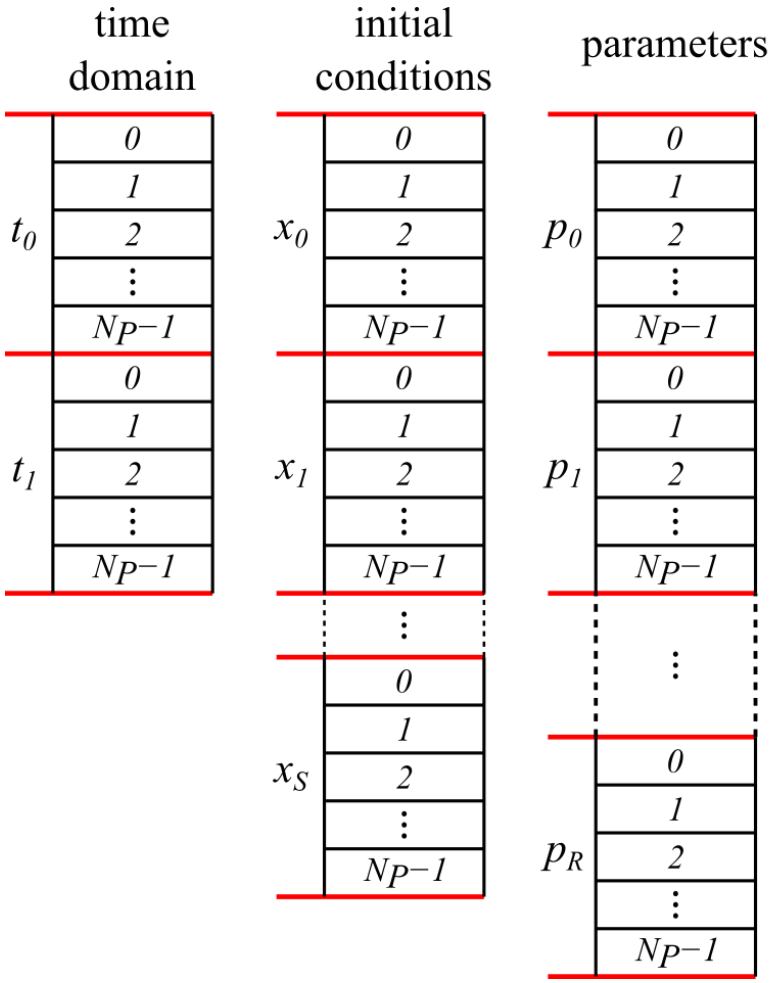
- *The main aims during the develop a general purpose parametric ODE solver:*
  - *Easily adopt for other systems*
  - *Easy set-up via user defined functions as similar as in MATLAB*

Solver: Main Concepts

- *The main concept during the develop a general purpose parametric ODE solver:*
  - *Do NOT store intermediate points (fast solver)*
  - *Flexible event handling (e.g. including impact dynamics)*
  - *Use of "Accessories" (e.g. maximum of solution)*

Solver: The Problem Pool

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



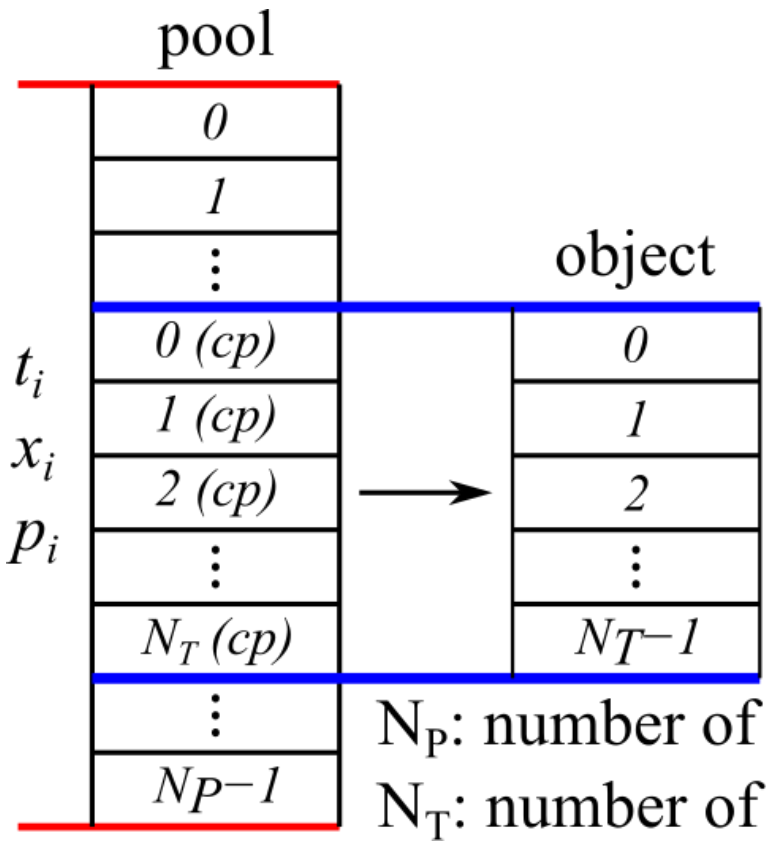
N_P : Number of problems in the pool
S: the size of a system
R: number of parameters

1 thread – 1 problem

Solver: The Solver Object

- *Create solver object and copy a portion of problems into its internal storages:*

```
ParametricODESolverConfiguration ConfigurationSystem;
     ConfigurationSystem.SystemDimension      = 2;
     ConfigurationSystem.NumberOfThreads      = 7680;
     ConfigurationSystem.NumberOfParameters   = 21;
     ConfigurationSystem.NumberOfEvents       = 1;
     ConfigurationSystem.NumberOfAccessories = 4;

ParametricODESolver ScanSystem(ConfigurationSystem);
```



N_P: number of problems in the pool

N_T: number of threads in the object

Solver: Perform Iterations

- *Define solver properties and perform simulation:*

```
SolverConfiguration SolverConfigurationSystem;
     SolverConfigurationSystem.BlockSize       = 64;
     SolverConfigurationSystem.InitialTimeStep = 1e-3;
     SolverConfigurationSystem.Solver          = RKCK45;

ScanSystem.Solve(SolverConfigurationSystem);
```

- *Only explicit schemes are available (yet):*
  - *Runge—Kutta—Kash—Carp (adaptive, orders 4-5)*
  - *4^{th} order Runge—Kutta (fix step size)*

Details: The Right Hand Side

- *A simple example (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*
  - *p2: excitation amplitude*

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

```
for (int i=0; i<NumberOfIterations; i++)
{
      ScanSystem.Solve(SolverConfigurationSystem);

      SAVE SOME PROPERTIES;
}
```

Details: The Right Hand Side

- *Definition of the right hand side:*

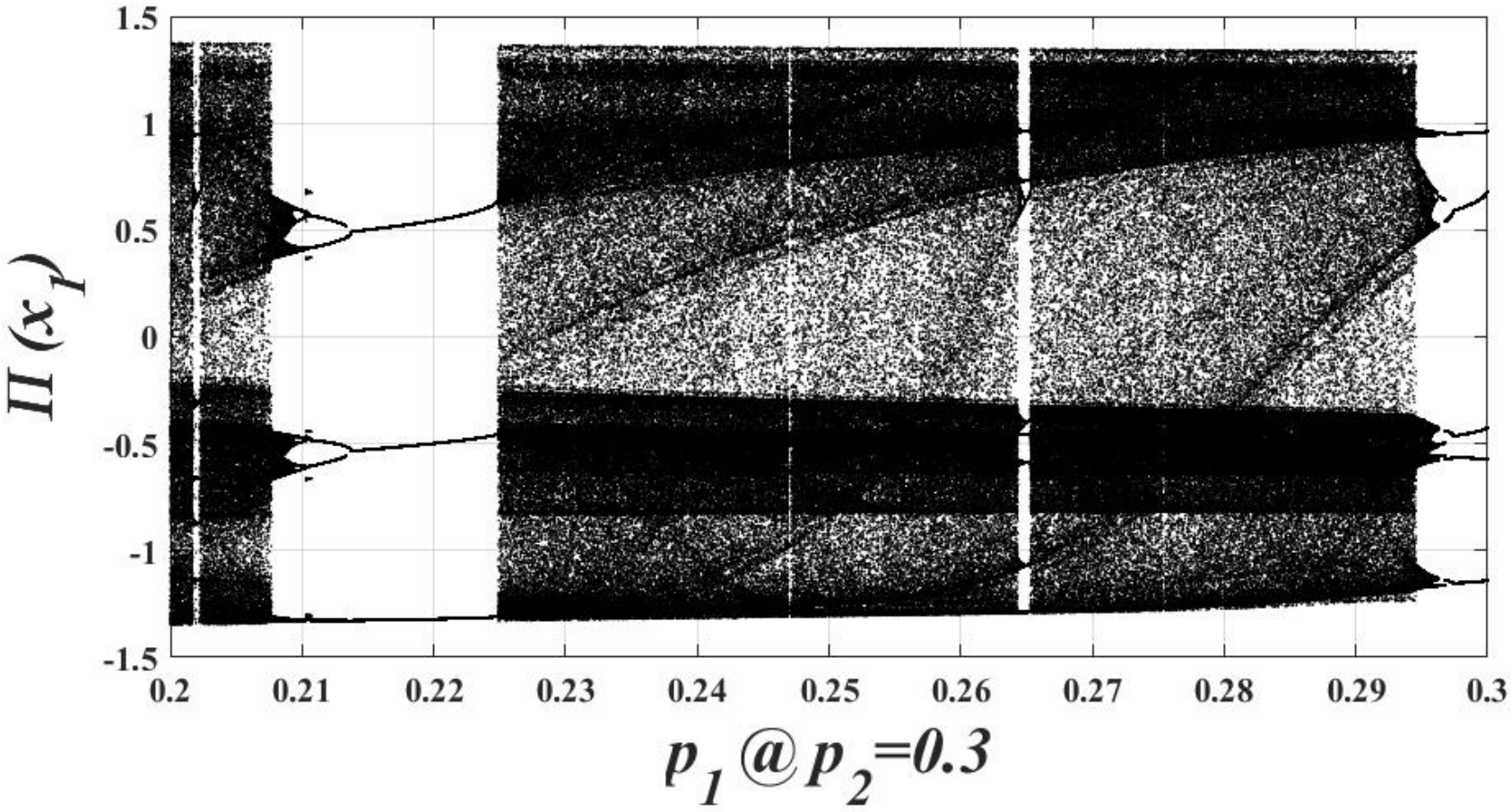
```
__device__ void ParametricODE_Solver_OdeFunction(double* RightHandSide, int idx, int NoT, double t, double*
StateVariable, double* Parameter)
{
	double x1 = StateVariable[idx + 0*NoT];
	double x2 = StateVariable[idx + 1*NoT];

	double p1 = Parameter[idx + 0*NoT];
	double p2 = Parameter[idx + 1*NoT];

	RightHandSide[idx + 0*NoT] = x2;
	RightHandSide[idx + 1*NoT] = x1 - x1*x1*x1 - p1*x2 + p2*cos(t);
}
```

- *User defined options:*

```
__device__ void ParametricODE_Solver_OdeProperties(double* RelativeTolerance, double* AbsoluteTolerance, double&
MaximumTimeStep, double& MinimumTimeStep, double& TimeStepGrowLimit, double& TimeStepShrinkLimit)
{
	RelativeTolerance[0] = 1e-10;
	RelativeTolerance[1] = 1e-10;

	AbsoluteTolerance[0] = 1e-10;
	AbsoluteTolerance[1] = 1e-10;

	MaximumTimeStep     = 1.0;
	MinimumTimeStep     = 1.0e-10;
	TimeStepGrowLimit   = 5.0;
	TimeStepShrinkLimit = 0.1;
}
```

Details: The Right Hand Side

- *A simple bifurcation diagram:*



Details: The Accessories

- *A more complex example (collapse of a dual-frequency driven gas bubble):*

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

$$x_1=R \qquad x_2=\dot{R}$$

- *Dual-frequency excitation:*

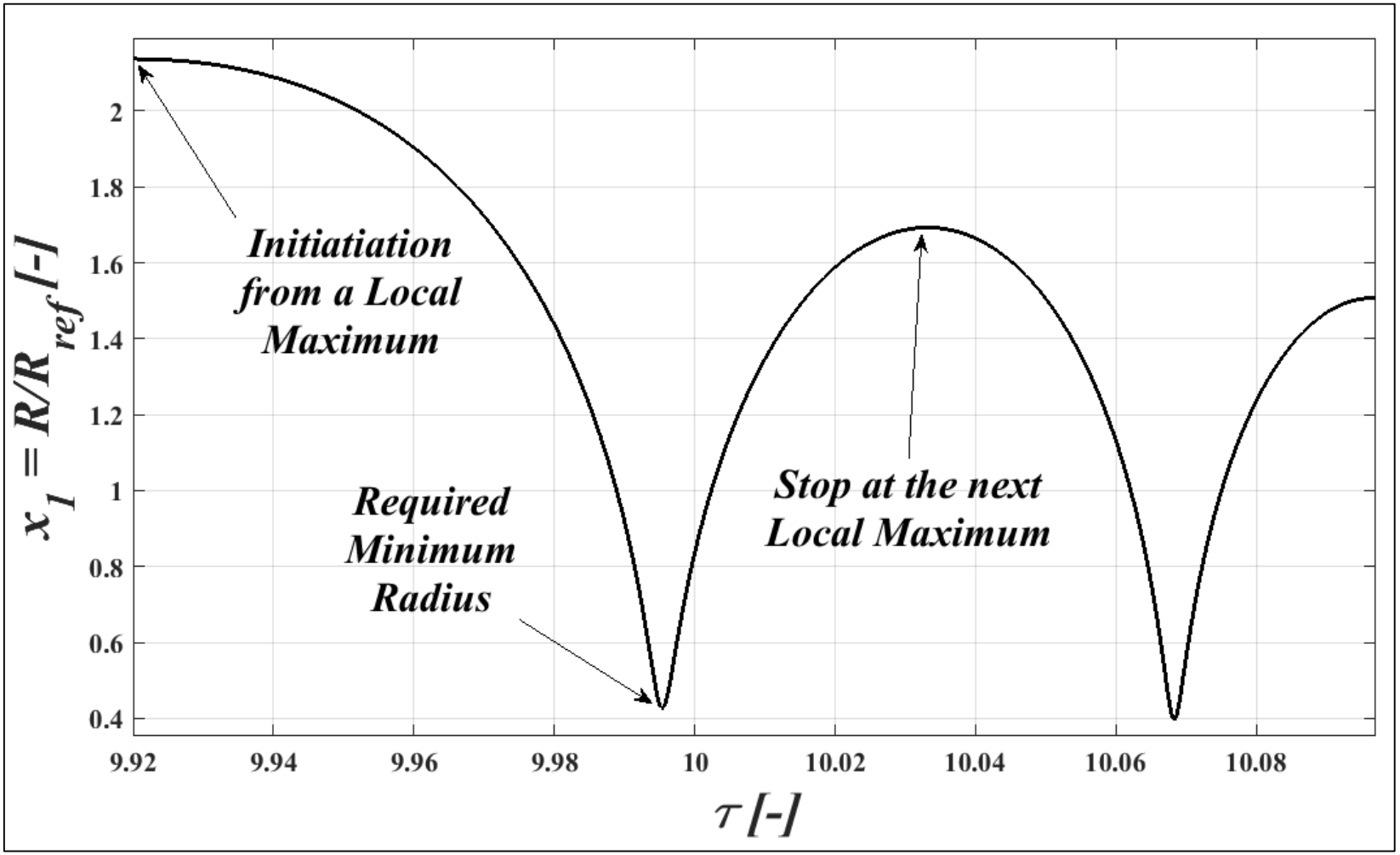
$$p_\infty(t)=P_{A1}\sin(\omega_1 t)+P_{A2}\sin(\omega_2 t+\Theta)$$

- *Investigated parameter space (resolution is 100):*

$$P_{A1},\ P_{A2},\ \omega_1,\ \omega_2,\ (\Theta=0)$$

Details: The Accessories

- *The collapse strength:*



Details: The Accessories

- *Initiation of accessories:*

```
__device__ void ParametricODE_Solver_Initialization(int idx, int NoT, double t, double* TimeDomain, double*
StateVariable, double* Parameter, double* Accessories)
{
	double x1 = StateVariable[idx + 0*NoT];

	Accessories[idx + 0*NoT] = x1;
	Accessories[idx + 1*NoT] = t;
	Accessories[idx + 2*NoT] = x1;
	Accessories[idx + 3*NoT] = t;
}
```

- *The accessories function:*

```
__device__ void ParametricODE_Solver_OrdinaryAccessories(double* Accessories, int idx, int NoT, double t, double*
StateVariable, double* Parameter)
{
	double x1 = StateVariable[idx + 0*NoT];

	if ( x1<Accessories[idx + 2*NoT] )
	{
		Accessories[idx + 2*NoT] = x1;
		Accessories[idx + 3*NoT] = t;
	}
}
```

Details: The Accessories

➢ *The event function:*

```
__device__ void ParametricODE_Solver_EventFunction(double* EventFunction, int idx, int NoT, double t, double*
StateVariable, double* Parameter)
{
	double x2 = StateVariable[idx + 1*NoT];

	EventFunction[idx + 0*NoT] = x2;
}
```

➢ *The event function properties:*

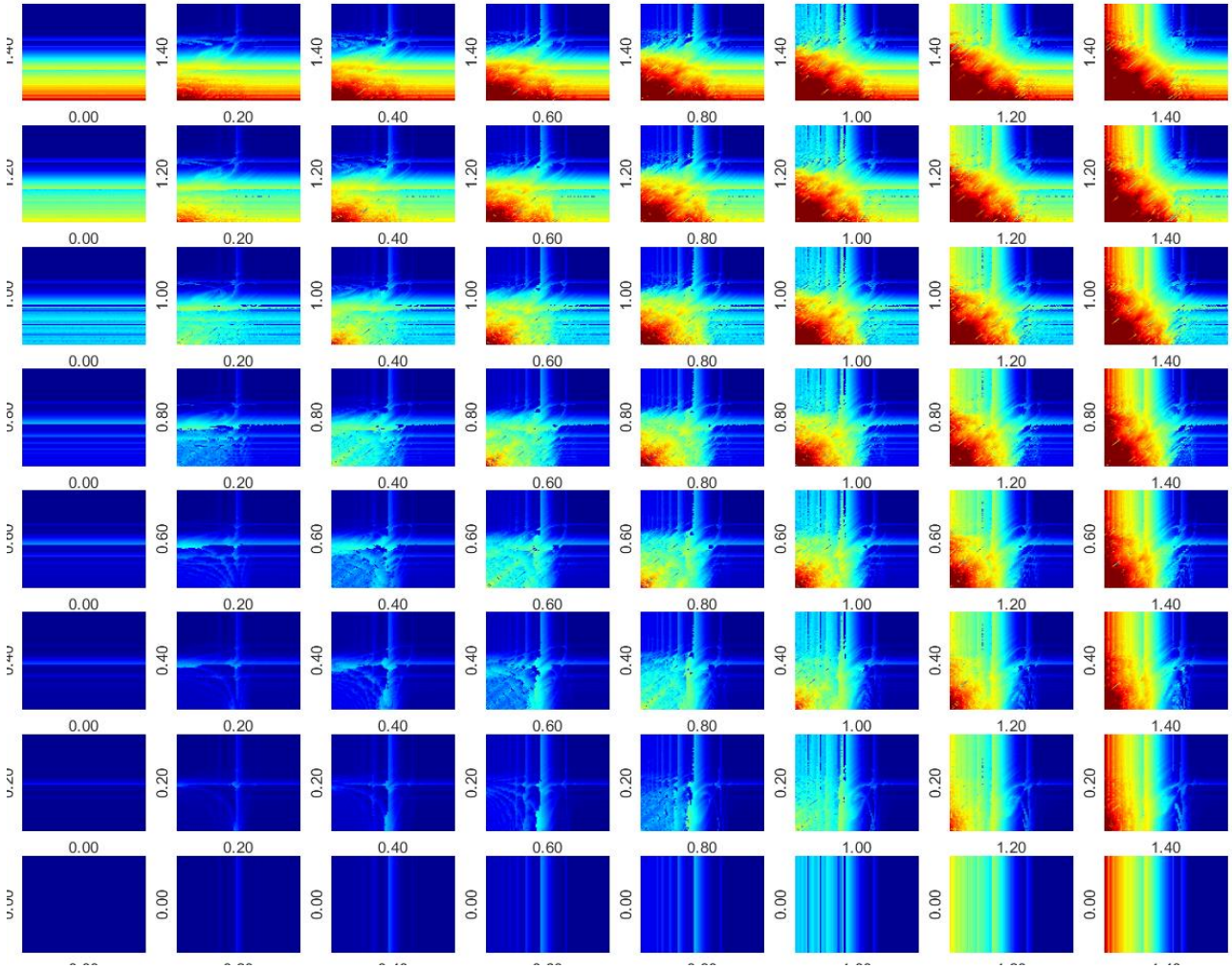
```
__device__ void ParametricODE_Solver_EventProperties(int* EventDirection, double* EventTolerance, int*
EventStopCondition, int& MaximumIterationForEquilibrium)
{
	EventDirection[0] = -1;

	EventTolerance[0] = 1e-6;

	EventStopCondition[0] = 1;

	MaximumIterationForEquilibrium = 50;
}
```

➢ *Finalization:*

```
__device__ void ParametricODE_Solver_Finalization(int idx, int NoT, double t, double* TimeDomain, double*
StateVariable, double* Parameter, double* Accessories)
{
	TimeDomain[idx + 0*NoT] = t;
}
```

Details: The Accessories

- *A 4D parameter scan of the collapse strength:*



Details: The Event Handling

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

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -\kappa x_2 - (x_1 + \delta) + x_3$$
$$\dot{x}_3 = \beta(q - x_1\sqrt{x_3})$$

- *The impact law:*

$$x_1^+ = x_1^- = 0$$
$$x_2^+ = -r x_2^-$$
$$x_3^+ = x_3^-$$

- *Parameter:*
  - *q: flow rate*
  - *r: coefficient of restitution*

Details: The Event Handling

- *The event functions:*

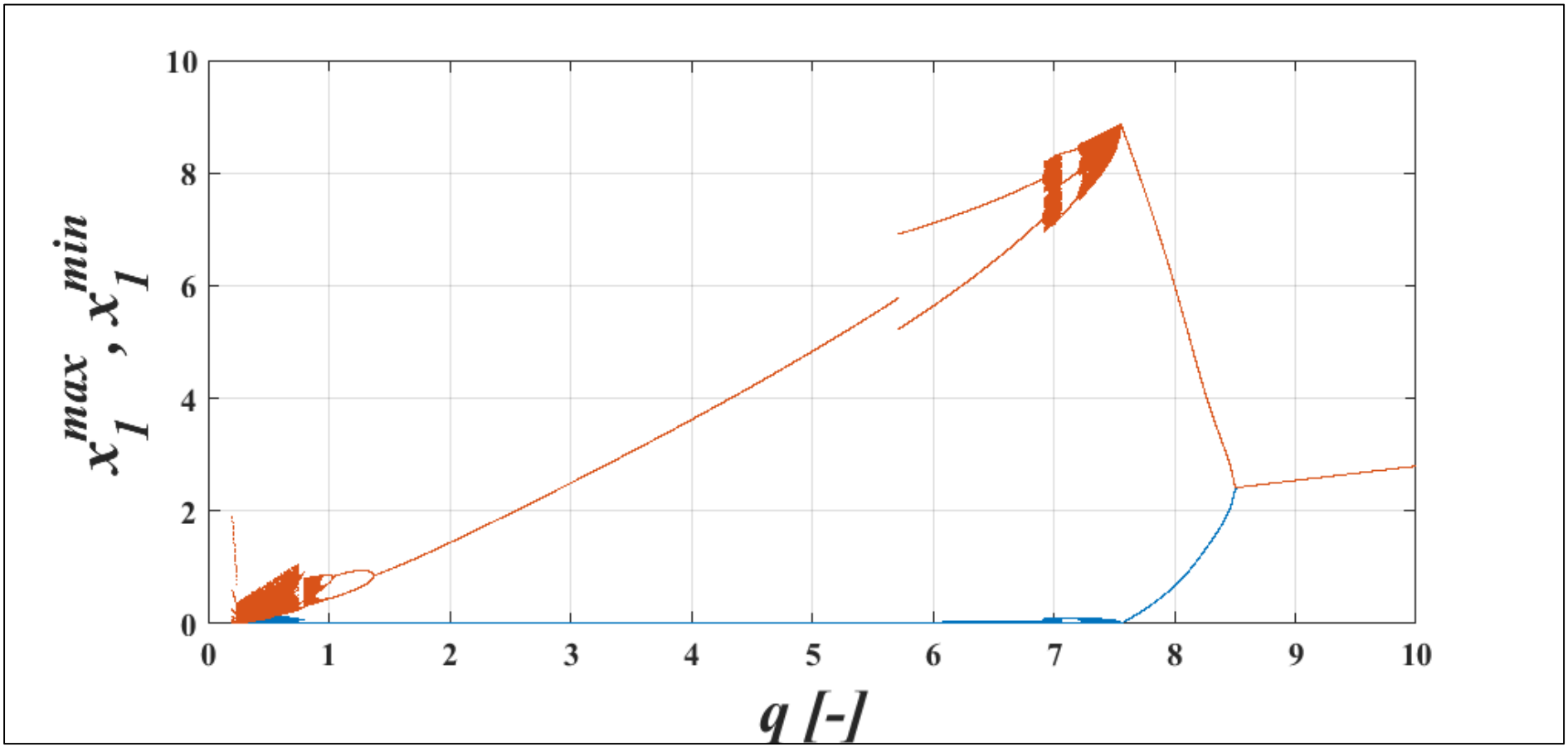
```
__device__ void ParametricODE_Solver_EventFunction(double* EventFunction, int idx, int NoT, double t, double*
StateVariable, double* Parameter)
{
	double x1 = StateVariable[idx + 0*NoT];
	double x2 = StateVariable[idx + 1*NoT];

	EventFunction[idx + 0*NoT] = x2; // Poincaré section
	EventFunction[idx + 1*NoT] = x1; // Impact detection
}
```

- *The event "action" function:*

```
__device__ void ParametricODE_Solver_EventActionFunction(int idx, int NoT, int EventIndex, int EventCounter,
double t, double* StateVariable, double* Parameter)
{
	double p5 = Parameter[idx + 4*NoT];

	if ( EventIndex == 1 )
	{
		StateVariable[idx + 1*NoT] = -p5 * StateVariable[idx + 1*NoT];
	}
}
```

Details: The Event Handling

- *The bifurcation diagram with impact:*



Summary

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

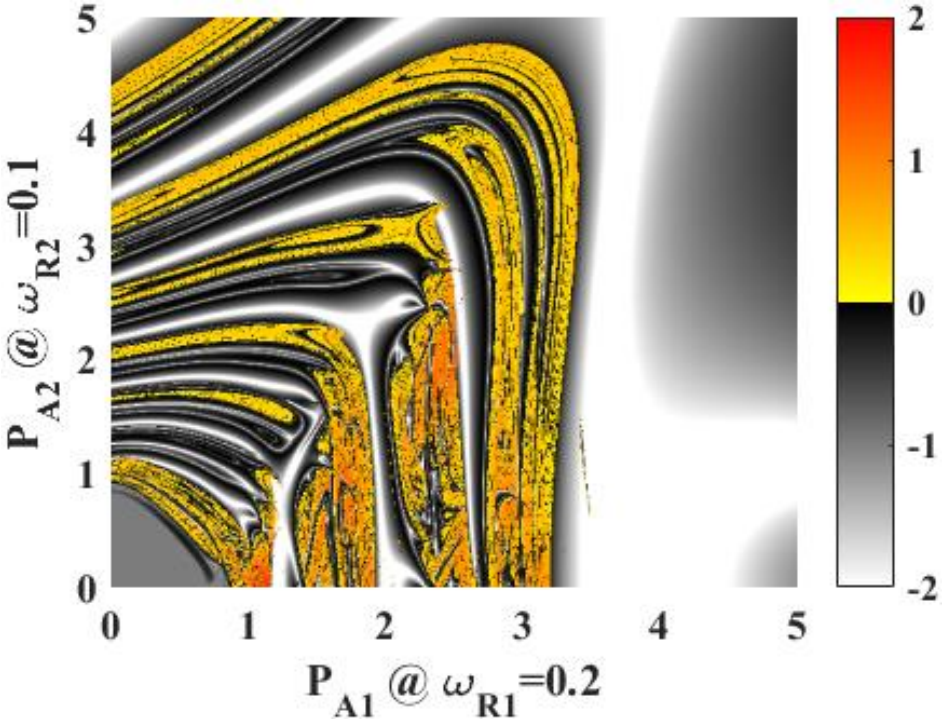
- *Possible developments:*
  - ***Make it more user friendly***
  - ***Include additional numerical schemes***
  - ***Implement efficient global/local couplings***

Thank you for your attention!

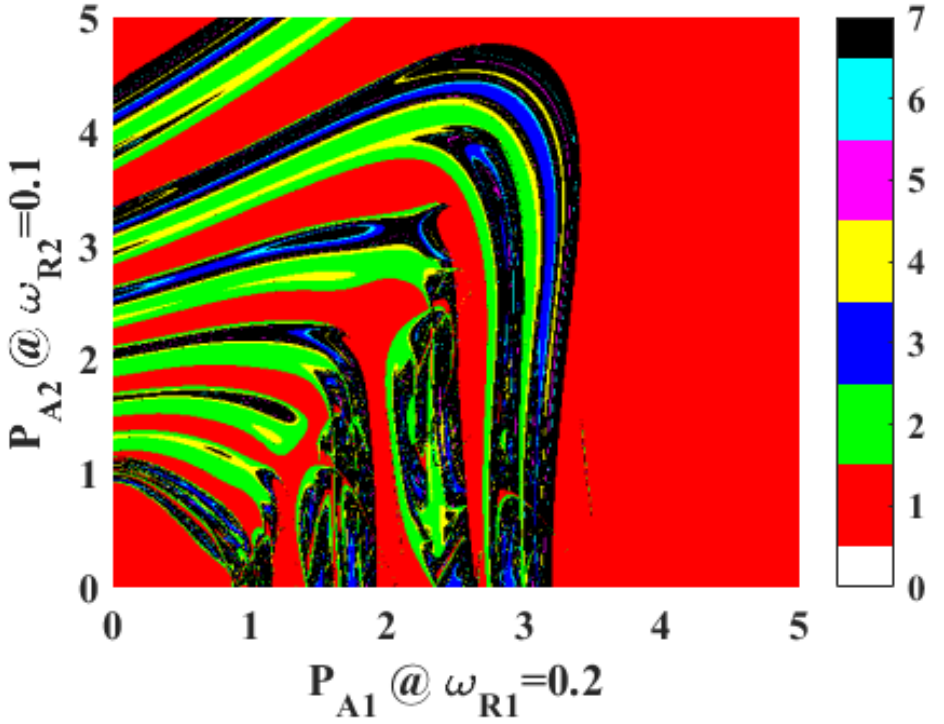
Example: Chaos Control

- *Chaotic and periodic solutions in a 2D parameter plane:*

- *Lyapunov exponent*

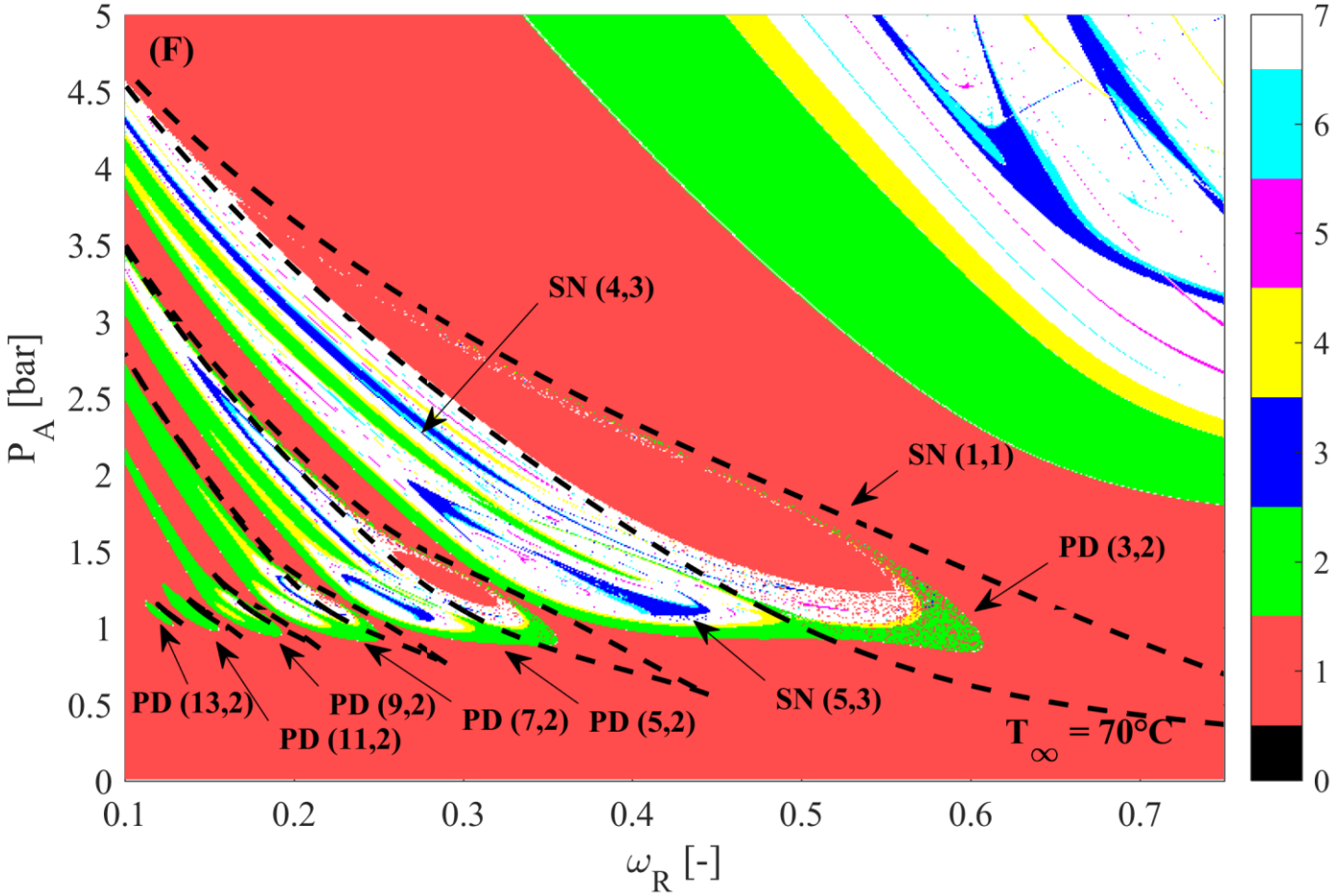


- *Period*



Example: Resonance Structure

- *Organization of harmonic resonances in a parameter plane:*



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;
}
```